MASTER EN SCIENCES INFORMATIQUES

Méthodes utilisées lors d'un essai du test de Turing

Cordier, Denis; Bastin, Véronique

Award date:
1998

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1997-1998

Méthodes utilisées lors d'un essai du « Test de Turing »

Mémoire réalisé par Véronique Bastin et Denis Cordier

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique.

Abstract

Chaque année, et ce depuis 1991, le Cambridge Centre for Behavioural Studies organise une compétition relative au « Test de Turing » : le *Loebner Prize*. Le principe de cette compétition est le suivant : un interrogateur converse avec un ordinateur ou un être humain. La suggestion de Turing était, que si l'interrogateur pouvait faire la différence entre les répliques de l'ordinateur et les répliques formulées par l'homme, alors l'ordinateur pense. Dans ce mémoire, nous présentons notre application qui a participé au *Loebner Prize* de 1998. Nous exposons les différentes méthodes implémentées qui ont favorisé un comportement humain de l'application. Pour terminer, nous présentons également d'autres techniques qui pourraient être utilisées en vue d'une amélioration future de ce programme.

Every year since 1991, the Cambridge Centre for Behavioural Studies has held a competition concerning the Turing Test, called the *Loebner Prize*. The principle of this competition is as follows : an interrogator receives responses from a computer and from a human being. The hypothesis of Turing was that, if we differentiate between a computer's reponses and a human being's, the computer can think. In this thesis, we present the application for the *Loebner Prize* contest 1998. We present implemented methods which favoured an application's human behaviour. Moreover, we also present other techniques which could be used for an improvment of this program.

Remerciements

Nous tenons à exprimer nos plus vifs remerciements à notre promoteur, le Père J. Berleur s.j. qui a rendu possible notre stage, en nous donnant l'occasion de travailler dans le domaine du traitement du langage naturel, ainsi que pour l'aide qu'il nous a apporté durant la rédaction de ce mémoire.

Nous remercions également David M. W. Powers, professeur à Flinders University of South Australia, pour les conseils qu'il nous a prodigué, ainsi qu'à la grande patience dont il a fait preuve lors de nos nombreuses questions. Nous ne devons pas oublier sa famille, Sue et Catherine, dont les encouragements nous ont été d'une aide précieuse.

Nous portons également une profonde reconnaissance aux professeurs de l'institut pour le bagage intellectuel qu'ils nous ont permis d'acquérir.

Tout d'abord, je remercie chaleureusement mes parents qui m'ont donné la possibilité de réaliser mes études et mon stage en Australie ainsi que pour leurs encouragements et leur affection Je porte également une profonde reconnaissance à ma marraine et mon oncle pour le soutien et la tendresse qu'ils m'ont procurée durant ces années d'étude.

Enfin, je n'oublie pas le reste de ma famille et mes amis qui m'ont continuellement encouragée. Des amis précieux qui m'ont apporté leur soutien et leur compréhension durant mes études et mon stage.

Véronique Bastin

Je remercie profondément mes parents, qui m'ont donné la possibilité de réaliser mes études et mon stage en Australie. Je remercie également toutes les personnes qui m'ont aidé, ou simplement soutenu durant cette période.

Denis Cordier

Table des matières

Introduction

Ce mémoire développe les travaux que nous avons menés dans le domaine du traitement du langage naturel, pendant notre stage à Flinders University of South Australia. Ce stage a été réalisé sous la supervision de David M. W. Powers, directeur du département d'intelligence artificielle, ancien président (1993-1997) de l'*ACL's SIG on Natural Languge Learning* et aujourd'hui membre de son International Advisory Commitee.

Parmi les diverses recherches de ce département, celles sur l'apprentissage du langage naturel, la perception de la parole et de la musique et la perception et traitement de la parole et de la vision ont retenu notre attention.

Le traitement du langage naturel étant pour nous un domaine jusqu'alors relativement inconnu, nous avons passé les premiers temps de notre stage à nous familiariser avec des sujets tels que l'apprentissage du langage par les enfants, les techniques utilisées en informatique pour acquérir ce langage ou l'utiliser comme, par exemple, l'apprentissage statistique du langage, ou encore la classification...

Un premier objectif nous fut alors assigné. David M. W. Powers nous a demandé de réaliser un programme participant au Loebner Prize (voir chapitre III.A.) qui devait se tenir en janvier 1998 à Sydney. Ce programme devait permettre d'imiter le mieux possible le comportement humain lors d'une conversation d'ordre général.

Une sélection préalable des participants était planifiée environ deux semaines plus tard, ce qui nous laissait peu de temps pour réaliser un prototype. Cette sélection était basée sur la qualité d'un script reprenant un extrait de conversation entre ce prototype et un être humain. La première version que nous avons du réaliser dans ce court laps de temps, comportait juste le « moteur principal » permettant de composer des conversations élémentaires. Elles portaient sur des sujets très limités.

Après avoir été sélectionnés, nous nous sommes interrogés sur la façon dont nous pourrions améliorer le comportement de l'application. Nous avons relevé deux tâches à réaliser :

- Améliorer et compléter le moteur principal en lui permettant de déterminer plus finement les sujets de conversation abordés par l'utilisateur du programme et, de ce fait, générer des commentaires plus adéquats dans le contexte envisagé.
- Introduire des techniques de correction orthographique. Cet ajout nous a été demandé par David M. W. Powers et nous a permis de nous initier aux techniques afférentes.

Les six semaines suivantes ont été consacrées à l'implémentation et aux tests de ces modules.

Durant cette même période, nous avons participé à l'organisation de la conférence *A*ustralian *N*atural *L*anguage *P*rocessing *F*ortnight (ANLPF)[1]. Notre participation nous imposait :

- La gestion des textes présentés lors de la conférence. Cette gestion nous demandait de réaliser les tâches suivantes :
  - ◊ Réception des textes et envoi des accusés de réception personnalisés aux participants.
  - ◊ Réception des évaluations de ces articles (système de cotation).
  - ◊ Création de scripts réalisés en Perl permettant de calculer d'une manière automatique la cotation finale de chaque article sur base des différentes évaluations. Cette cotation permettait de décider de l'acceptation ou du rejet du texte concerné lors du déroulement de la conférence.

[1] *Voir annexe C*

- L'édition de l'ouvrage reprenant l'ensemble des articles présentés lors de la conférence. Cette édition consistait en :
    - ◊ Standardisation des différents textes réceptionnés par modification de leur code en PostScript.
    - ◊ Insertion des bas de pages relatifs à chaque article.
    - ◊ Confection du document final en vue de l'impression.

Cette participation à l'organisation de la conférence ne sera plus abordée par la suite. Nous l'avons juste indiquée pour permettre de préciser la suite continue des tâches effectuées. Un article relatif à l'architecture finale de notre application[2] a été publié dans le cadre de cette conférence. Une présentation en a été faite lors d'un séminaire à Flinders University, une autre lors d'un *workshop* à Sydney.

Nous avons assisté à l'entièreté de la conférence *ANLPF*, directement précédée par la compétition (Loebner Prize) à laquelle nous avons participé.

Qu'en est-il de la structure de notre mémoire ? Le chapitre premier introduit les différentes disciplines en relation avec les principes utilisés lors du développement de notre application ou de son évolution future. Le chapitre deux présente trois techniques liées à l'apprentissage du langage par les machines. Ce chapitre nous permet d'introduire les perspectives d'évolution du programme réalisé, dont certaines nécessitent l'utilisation des techniques présentées. Le chapitre trois, quant à lui, présente l'architecture générale de notre application, ainsi qu'un outil, utilisé par celle-ci, permettant de déterminer des relations sémantiques entre des mots du langage. Le dernier chapitre permet de faire le lien entre le chapitre trois relatif à notre application actuelle et le chapitre deux relatant trois techniques pouvant être utilisées pour modéliser l'acquisition du langage par les machines. Ce lien se traduit par l'utilisation des techniques exposées au chapitre deux dans une architecture « future » de notre application. Enfin nous terminons par une conclusion dans laquelle nous rassemblons les richesses retirées de notre stage, mais aussi les inconvénients dus à notre ignorance du domaine.

---

[2] *Voir annexe B*

Chapitre I
Les sciences cognitives

A. Introduction

L'intelligence artificielle est devenue une discipline à elle seule. Il y a une vingtaine d'années, les chercheurs d'autres disciplines ressentent le besoin de s'impliquer dans cette nouvelle science. Tout d'abord, les psychologues qui sont de plus en plus concernés par les renouvellements que l'intelligence artificielle suscite dans leur discipline. Ensuite, les linguistes qui vont entamer de nouvelles explorations dans le domaine de la correspondance du syntaxique et du sémantique, de la forme et du sens du système de symboles. Ces deux disciplines (psychologie et linguistique) en plus des neurosciences, de l'intelligence artificielle et des sciences sociales ont formé un groupe constitutif des sciences de la cognition. (**[Lemoigne86]**)

Ces disciplines ainsi que les interactions entre elles sont reprises dans un schéma développé par la Fondation A.P. Sloan (**[Lemoigne86]**) :

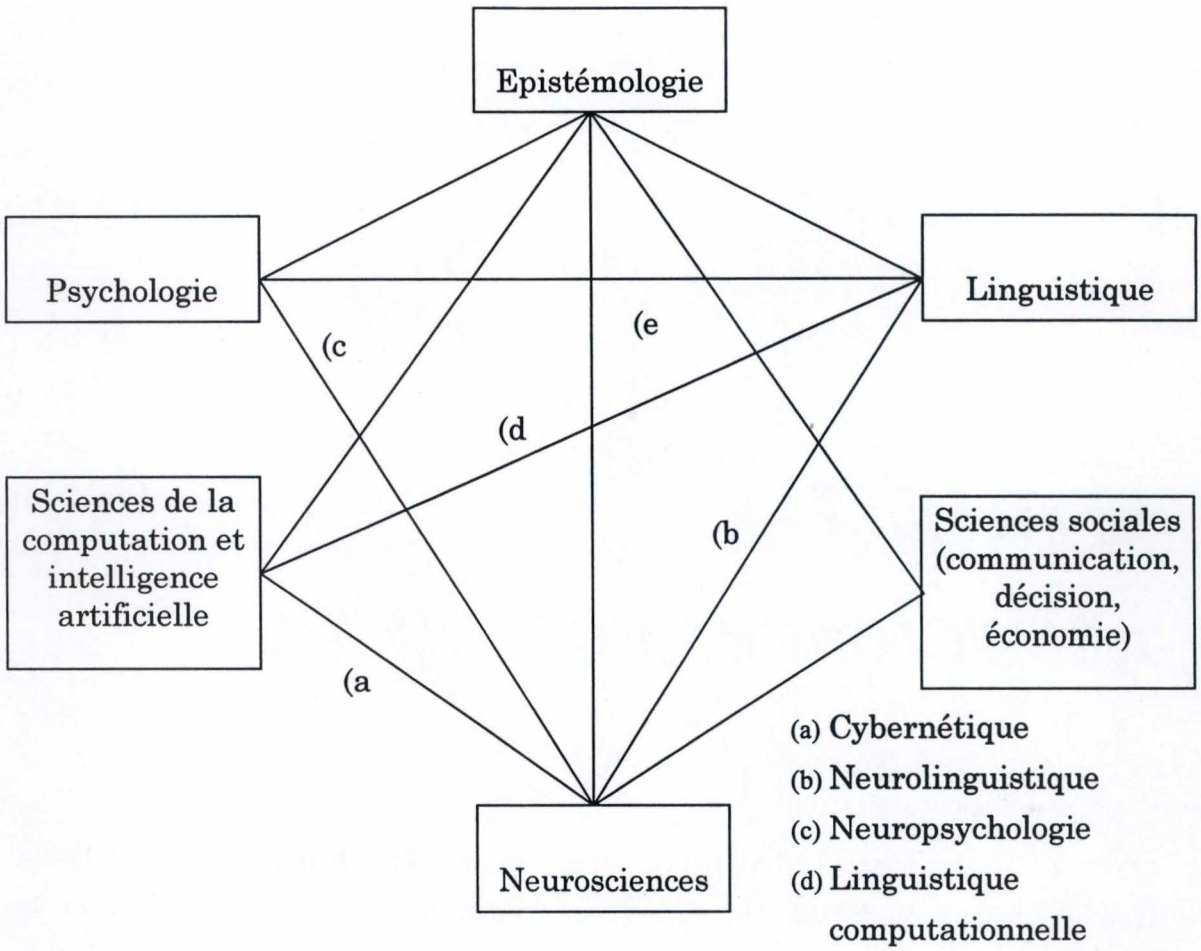


Figure 1-1 Le réseau constitutif des "Sciences de la Cognition"

La suite de ce chapitre est consacrée à une exploration un peu plus détaillée de quatre sciences reprises dans ce schéma, à savoir l'intelligence artificielle, la linguistique, les neurosciences et la psychologie cognitive. Nous avons choisi d'explorer ces quatre disciplines pour mettre en évidence leur apport dans de nombreuses applications relatives au traitement du langage naturel.

B. La psychologie cognitive

B.1. Introduction

Les psychologues cognitifs formulent et testent des théories à propos de l'esprit humain et du comportement. Ils voient, comme les scientifiques cognitifs, l'esprit humain comme un remarquable système de traitement de l'information. La psychologie cognitive essaye de déterminer des hypothèses selon lesquelles les connaissances sont représentées dans le cerveau humain, et quels sont les mécanismes permettant de manipuler cette connaissance. De tels mécanismes jouent un rôle déterminant dans les capacités des machines ou organismes vivants. On peut nommer cet ensemble de mécanismes une *architecture fonctionnelle*.

La psychologie cognitive va donc essayer de déterminer cette architecture fonctionnelle, sans s'occuper du fonctionnement réel du système nerveux, des neurones...

Dans ce chapitre, nous nous limitons à introduire certains domaines de la psychologie cognitive, permettant d'introduire un outil que nous présentons dans le chapitre III : WordNet. Pour cela, nous exposons certaines hypothèses sur la manière dont les connaissances pourraient être représentées chez les hommes, ainsi que certains mécanismes qui permettraient de manipuler cette connaissance.

Plusieurs théories de la représentation des connaissances chez les êtres humains ont été développées par la psychologie cognitive. Expliquons brièvement deux théories possibles de cette représentation.

B.2. Les façons de représenter de la connaissance

La psychologie cognitive a émis différentes hypothèses de représentation de la connaissance dans le cerveau humain. **[Stillings et al. 87a]** définit deux moyens de représenter la connaissance : la représentation propositionnelle et la représentation schématique.

La **représentation propositionnelle** émet l'hypothèse que notre connaissance est représentée sous une forme de propositions simples, reliées

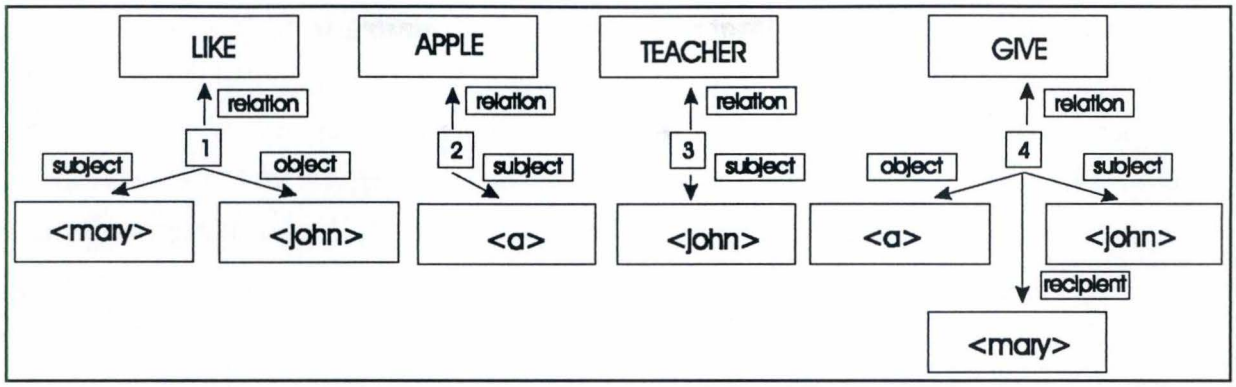
entre elles sémantiquement. Cette représentation serait inconsciente et traduite en langage naturel au moment où nous parlons. Bien que rien ne soit prouvé quant à cette possibilité de représentation du langage, certains faits sont en faveur de cette dernière. Par exemple, cette hypothèse expliquerait l'expression bien connue « j'ai le mot sur le bout de la langue », où on a le mot à l'esprit, mais on ne parvient pas à l'exprimer. De plus, nous générons assez facilement des mots par rapport à un concept (table, chaise...), et sommes beaucoup moins aptes à la découverte des mots justes d'une définition. Ce phénomène suggère que la définition des mots serait codée dans une représentation interne, associée au mot en question. Un dernier argument est la constatation que les jeunes enfants parviennent à acquérir des concepts en dépit du fait qu'ils ne maîtrisent pas totalement un langage.

Une expérience réalisée par **[Sachs67]** ajoute encore un poids aux arguments déjà avancés. Il faisait écouter quelques phrases indépendantes. Les passages étaient interrompus, et les auteurs de l'expérience présentaient aux sujets un test de mémoire. Ce test consistait à présenter aux sujets une phrase, et leur demander si elle appartenait au passage lu. Les différentes phrases de test sont inscrites ci-dessous, la phrase correcte est la première.

1. He sent a letter about it to Galileo, the great Italien scientist.
2. He sent Galileo, the great Italien scientist, a letter about it.
3. A letter about it was sent to Galileo, the great Italian scientist.
4. Galileo, the great Italien scientist, sent him a letter about it.

Les phrases deux et trois ont la même signification que la première. Les sujets ont rarement fait l'erreur de prétendre que la phrase 4 apparaissait dans le texte. Par contre, les phrases 2 et 3 ont été souvent citées. Une explication de ce phénomène serait que la connaissance est codée sous une forme non linguistique, ce qui ferait que plusieurs phrases, de même signification, sont codées de la même manière.

Donnons un exemple de représentation propositionnelle des phrases « *Mary likes John* » et « *Mary likes the teacher who gave her an apple* ».



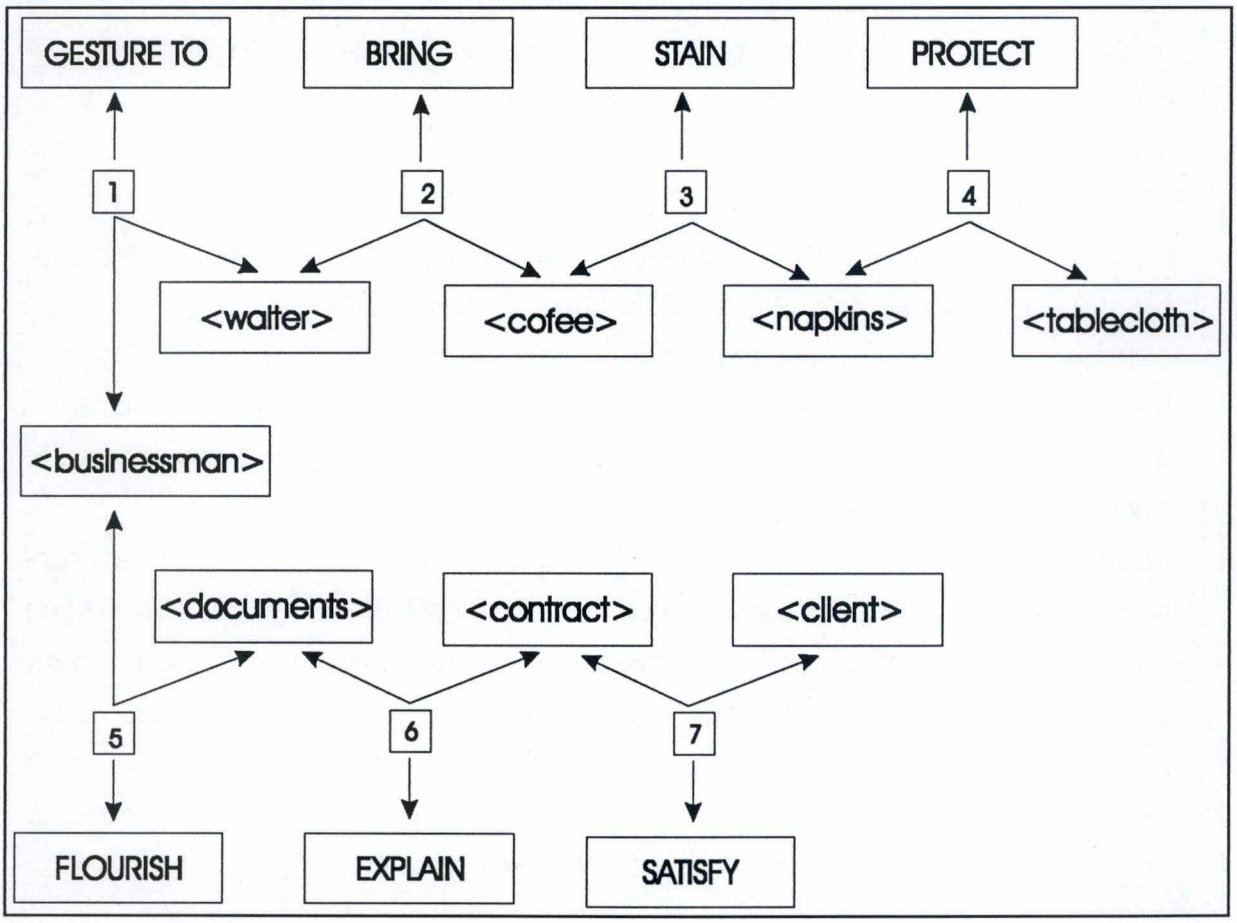
Ces différents schémas vont alors être reliés en un seul schéma plus complexe, un *réseau de propositions*. La relation un peu spéciale APPLE ne fait qu'assigner une valeur à une sorte de variable qui sera utilisée par la suite. Cette variable <a> permet de représenter l'objet particulier (pomme) que donne John à Mary.

La psychologie cognitive a encore franchi une étape et avance que les réseaux propositionnels peuvent être considérés comme un modèle de la mémoire déclarative à long terme. L'hypothèse de base de cette théorie est que à chaque moment, chaque noeud du réseau est à un certain *niveau d'activation*, et ce niveau se répand aux autres noeuds via les liens. Tous les noeuds pour lesquels le niveau d'activation est assez élevé forment un sous-réseau qui est accessible à notre partie consciente, ce qui permet de retrouver l'information ainsi sélectionnée. De nouveau, la psychologie cognitive utilise certaines hypothèses, par exemple celle comme quoi tous les noeuds seraient activés au même moment, ce qui conduirait les niveaux d'activation à se « répandre » en parallèle. Une seconde hypothèse est que le niveau d'activation est divisé entre tous les liens arrivant à un noeud. Cela implique que le niveau d'activation d'un noeud, lorsqu'il se répand à travers le réseau, diminue rapidement lors de ses divisions successives à chaque noeud rencontré. La théorie suppose aussi que l'activation d'un noeud décroît rapidement avec le temps. Enfin, on suppose que les noeuds et les liens peuvent avoir différentes capacités d'activation. On peut tout de suite faire le rapprochement avec la programmation parallèle ainsi qu'avec la modélisation des réseaux neuronaux, utilisée en informatique.

De nouveau, une expérience a été menée par **[McKoon et al.]**, qui a conforté l'hypothèse des réseaux propositionnels. Les sujets ont lu deux courts paragraphes avec de simples structures propositionnelles. Ensuite on

leur citait des mots et on leur demandait de répondre le plus vite possible s'ils pensaient que le mot était présent dans une des histoires ou pas. Les chercheurs ont proposé trois types d'essais : les essais avec amorçage, où le mot-test était précédé par un mot-test dans la même histoire, les essais sans amorçage où le mot-test était précédé d'un mot-test de l'autre histoire. Dans le dernier type d'essai le mot-test n'apparaissait dans aucune des histoires. Il ont constaté que les réponses aux tests avec amorçage étaient plus rapides que les réponses aux tests sans amorçage. La théorie des réseaux prédit cette constatation parce qu'il y a toujours des chemins entre mots d'une même histoire; donc on a des noeuds déjà activés. Voici un des paragraphes proposé par les scientifiques et son réseau associé :

The businessman gestured to a waiter. The waiter brought coffee. The cofee stained the napkins. The napkins protected the tablecloth. The businessman flourished documents. The documents explained a contract. The contract satisfied the client.



La représentation propositionnelle est assez simple, mais est largement incomplète. En effet, cette théorie n'apporte aucune information

sémantique. Par exemple, elle n'explique pas comment on peut savoir ce qu'est un document, ni à quoi il sert...). **[Stilling et al. 87b]** introduit alors une autre théorie, de la représentation, complétant la représentation propositionnelle et permettant d'obtenir une information sémantique.

La **représentation schématique** permet de prendre en compte une certaine « connaissance du monde », qui n'est pas prise en compte dans l'hypothèse précédente. Par exemple, pour comprendre la relation APPLE, il est nécessaire de savoir qu'une pomme est un fruit. On doit bien sûr faire apparaître cette propriété pour rester cohérent. On doit posséder le *concept* général de APPLE dans notre esprit. L'ensemble des caractéristiques générales relatives à APPLE s'appelle un *schéma*. Un schéma est donc une abstraction qui permet à certains objets ou événements d'être assignés à certaines *catégories*.

Il semblerait que l'ensemble des objets (classes des noms communs) est situé dans un endroit précis du cerveau. De plus, cet ensemble de noms serait organisé hiérarchiquement dans la mémoire humaine. A nouveau, il existe certains indices permettant de poser ces hypothèses.

Ceux-ci viennent d'observations réalisées sur des patients, dont une partie de l'hémisphère gauche du cerveau a été endommagée. Une grande partie de ces patients souffrent généralement d'un déficit de la capacité de nommer les objets **[Caramazza et al. 78]**, ce qui conforte l'hypothèse que la mémoire des noms serait déconnectée du reste de la mémoire lexicale du cerveau. Toutefois, les symptômes ne sont pas généralisés à tous les patients, et cet argument n'est donc pas « infaillible ».

La première personne à affirmer que les noms sont organisés hiérarchiquement dans le cerveau a été Quillian **[Quillian67]** **[Quillian68]**. Les tests expérimentaux de la proposition de Quillian ont été rapportés dans un article de **[Collins et al. 69]**, qui affirment que les temps de réaction peuvent être utilisés pour indiquer le nombre de niveaux hiérarchiques séparant deux significations. Les expériences ont révélé qu'il prenait moins de temps pour répondre *oui* à la proposition « *Un canari sait chanter* » qu'à la proposition « *Un canari sait voler* », et encore plus de temps pour analyser « *Un canari a de la peau* ». Les scientifiques supposent donc que « *can sing* » est une caractéristique de *canari*, « *can fly* » est une

caractéristique de *bird*, et « *has skin* » est une caractéristique de *animal*, ce qui explique les différents temps de réponse.

La plupart des chercheurs en psycholinguistique sont d'accord sur le fait que les noms communs sont organisés hiérarchiquement dans la mémoire sémantique mais ne savent pas si l'information générique est héritée ou stockée avec redondance **[Miller et al. 93]**.

Cette notion de schéma résout le problème d'absence de sémantique qui caractérise la représentation propositionnelle, mais ne convient pas pour représenter l'entièreté des types de connaissances possibles. A cette fin, **[Stillings et al. 87b]** introduit d'autres concepts.

B.3. Processus automatique versus processus contrôlé

Après avoir défini une manière de représenter la connaissance, examinons les façons de traiter nos informations. Il existe des *procédures* permettant de manipuler la connaissance stockée. Les chercheurs en psychologie cognitive ont aussi développé des théories qui pourraient s'appliquer au cerveau humain. Un point très important est la notion de *processus automatique* et de *processus contrôlé*. Pour expliquer cette notion assez simple, un exemple suffira.

Lorsqu'on apprend à conduire une voiture, par exemple, on doit penser aux moments où il faut changer de vitesse, mettre les clignotants, au fur et à mesure que l'on acquiert de l'expérience, on ne pense plus, le processus devient automatique. Cette notion est présente entre autres lors de *l'acquisition des habiletés (Skill Learning)*.

B.4. L'acquisition des habiletés

Etant donné que le langage naturel est une caractéristique de l'homme, il nous a paru nécessaire de consacrer un paragraphe à ce sujet. De nombreux théoriciens ont tenté de diviser le processus d'apprentissage des habiletés en étapes qui caractérisent une transition entre le processus contrôlé et le processus automatique. Leurs conclusions sont assez similaires, et ils proposent de diviser le processus d'acquisition en trois étapes **[Stillings et al. 87b]**:

- Le **stade interprétatif** où on essaye de réaliser une tâche par essais successifs. En même temps on stocke la connaissance nécessaire à la réalisation de cette tâche. Au fur et à mesure des essais, on passe progressivement au stade suivant. On fait ici souvent appel à une *mémoire de travail*, qui a une capacité limitée.
- Le **stade compilé** est une période pendant laquelle les connaissances acquises sont « compilées » en une procédure qui est spécifique à la réalisation de la tâche (Skill). **[Anderson83]** conforte cette hypothèse par son observation des étudiants apprenant des aptitudes complexes : la résolution de problèmes de géométrie Euclidienne. Anderson observe le fait qu'au début, les étudiants font appel à la connaissance venant directement de leur livre, ou à celle qui se trouve dans leur mémoire à court terme . Au fur et à mesure du temps, l'étudiant va pouvoir résoudre des problèmes de ce type en faisant de moins en moins appel à la mémoire à court terme mais en faisant appel à la théorie qui, elle se trouve dans la mémoire à long terme. Finalement, l'étudiant va pouvoir résoudra les problèmes de manière pratiquement automatique, sans nécessairement faire appel aux notions théoriques. Le stade de compilation est alors terminé.
- Le **stade automatique** est finalement observé, où les procédures deviennent encore plus automatisées, on ne pense plus à ce qu'on fait, c'est « machinal ».

La psychologie cognitive comprend encore beaucoup d'autres sujets non abordés ici, comme la vision, le raisonnement. Nous n'avons voulu évoquer ici que ce qui permettait d'introduire le mieux possible le sujet de notre mémoire : *Méthodes utilisées lors d'un essai du « Test de Turing ».*

C. L'intelligence artificielle

C.1. Introduction

Suivant les recherches entreprises, J. Searle a proposé de distinguer deux approches de l'intelligence artificielle : l'intelligence artificielle forte, qui essaye de réaliser les tâches cognitives du cerveau humain, tout en essayant de comprendre son fonctionnement réel, et l'intelligence artificielle

faible qui s'efforce de trouver des procédés (algorithmes) permettant de simuler le comportement cognitif humain, mais sans avoir pour but d'en comprendre le fonctionnement.

[Berleur91] insiste, dans ce sens, sur l'importance de distinguer le mode de performance du mode de simulation, et de ne pas se tromper sur les objectifs de l'intelligence artificielle :

« L'IA vise-t elle, en effet, la simulation, la compréhension et la reproduction des mécanismes de l'homme, entre autres ses mécanismes langagiers ou se contente-t-elle d'accomplir des tâches à travers des traitements computo-symboliques ? »

Il existe de nombreuses définitions de l'intelligence artificielle. **[Lemoigne86]** définit l'intelligence artificielle comme la *« science des machines capables de manifester des comportements intelligents »*. **[Allsch93]** donne une définition légèrement différente : *«l'intelligence artificielle a pour but de faire exécuter par l'ordinateur des tâches pour lesquelles l'homme, dans un contexte donné, est aujourd'hui meilleur que la machine »*. On remarque que ces deux définitions englobent l'intelligence artificielle forte et faible. Les chercheurs en intelligence artificielle essayent de réaliser des programmes qui font ressortir des caractéristiques de l'intelligence humaine. Cela nécessite donc que les représentations et les processus décrits dans les théories soient étudiés en profondeur, pour pouvoir les implémenter. Un programme fonctionnant mal à un moment donné pourrait être un indice d'une erreur dans la théorie correspondante. On peut déjà faire ici la liaison assez simplement entre la psychologie cognitive, qui établit des théories et émet des hypothèses quant aux fonctions du cerveau, et l'intelligence artificielle qui cherche à implémenter ces théories et tester leur validité. Exposons brièvement différents thèmes de l'intelligence artificielle, largement utilisés en traitement du langage naturel.

C.2. Représentation de la connaissance

Nous avons déjà abordé ce sujet en traitant de la psychologie cognitive. Il faut évidemment pouvoir représenter la connaissance nécessaire aux traitements effectués par la machine. Par exemple, dans le traitement du langage naturel, il faut déterminer une manière de représenter le langage naturel. Cette connaissance doit pouvoir être utilisée,

manipulée, ou encore modifiée par la machine, à l'aide de processus adéquats. Ce terme de « représentation de la connaissance » implique plusieurs questions telles que

Quelle est la connaissance impliquée dans la réalisation de la tâche, ses types, structure et organisation?

Comment cette connaissance peut-elle être représentée dans l'ordinateur?

Quelle sorte de connaissance est rendue explicite par la représentation? Qu'est ce qui peut être implicite? Quelle sorte de connaissance peut-on représenter?

Comment la connaissance peut-elle être acquise ou modifiée?

C.3. Recherche et contrôle

Ces deux termes sont bien connus en intelligence artificielle, la *recherche* consiste à explorer les différentes alternatives d'un problème Des recherches heuristiques sont bien sûr possibles, on ne recherche alors que dans une partie de l'arbre des solutions. Le *contrôle* permet de gérer les différents processus dans un système, en vérifiant s'ils peuvent utiliser les ressources auxquelles ils tentent d'accéder.

C.4. L'apprentissage

De nombreux livres ont été consacrés à l'apprentissage. En intelligence artificielle, par apprentissage, on entend la manière dont un système « devient meilleur », c'est à dire comment sa base de connaissances est élargie. Cela peut signifier par exemple l'ajout de nouveaux faits à une base existante. Voici quelques questions importantes que toute personne travaillant dans ce domaine se pose :

Quelles capacités de connaissance le système d'apprentissage possède-t-il?

Comment les nouvelles connaissances et capacités sont elles apprises et stockées dans celles déjà existantes?

Quel est le rôle du professeur?

Quel est le rôle des exemples ou des expériences présentés au système d'apprentissage?

Nous nous sommes limités à décrire des « principes d'applications » typiques à l'intelligence artificielle. Il est en effet inutile de chercher à exposer exhaustivement toutes les techniques disponibles, celles-ci évoluant très rapidement. **[Allsch93]** énonce une liste de problèmes typiques de l'intelligence artificielle :

- La résolution de problèmes de façon générale
- La compréhension du langage naturel
- La reconnaissance de scènes visuelles
- L'étude de la fiabilité des opérateurs humains
- L'apprentissage
- Le modèle cognitif du raton-laveur etc...

D. Linguistique

D.1. Introduction

Le schéma de la Sloan Foundation continue dans les constituants de base des sciences de la cognition : la linguistique. L'application que nous développerons a pour objectif d'analyser une phrase (en langage naturel) et de formuler une réponse cohérente à cette phrase.

Qu'est-ce que la linguistique ? **[Ency9]** considère la linguistique comme l'étude scientifique du langage, tel qu'on peut l'appréhender à travers les langues naturelles (français, anglais, ...). **[Vihman96]** a émis différentes propriétés pouvant caractériser des systèmes de communication dont le langage. Nous avons choisi de conserver les termes anglais pour éviter toute confusion :

- ***Modality*** : le langage humain peut être exprimé dans un de ces deux modes d'expression :
  - ◊ vocal/auditif
  - ◊ gestuel/visuel

- ***Interchangeability*** : la personne qui reçoit et la personne qui transmet peuvent s'échanger leurs rôles dans la communication.
- ***Complete feedback*** : la personne qui transmet reçoit également.
- ***Specialization*** : la production de paroles ne sert pas de fonction biologique mais est utilisé uniquement pour la communication.
- ***Semanticity*** : les signaux linguistiques sont associés à des significations spécifiques.
- ***Discreteness*** : les langages utilisent un répertoire de sons et d'unités de signification.

Nous pouvons constater que les langages humains possèdent ces propriétés, contrairement à certains autres systèmes comme le langage des chimpanzés ou de certains insectes (la danse des abeilles, ...).

Des recherches menées dans le domaine de la linguistique, nous retenons deux théories les plus souvent reprises dans la littérature utilisée (**[Ency11], [Ducrot et al. 72]**) à savoir la théorie structuraliste distributionnelle et la théorie linguistique selon Chomsky. Dès 1950, la théorie structuraliste émerge. Elle a pour objectif d'isoler les éléments constitutifs d'un corpus (la langue naturelle) et de les classifier en un système. Le but est d'écrire des « formules » de phrases grâce à ces classes. Pourtant, en 1957, Chomsky réfute cette théorie. Celui-ci a pu démontrer que cette classification était impuissante à démontrer certains phénomènes linguistiques. Par exemple, la phrase « *la critique de Chomsky* » recouvre deux faits différents : « Chomsky critique quelqu'un » et « Chomsky est critiqué par quelqu'un ». De plus, pour Chomsky, cette perspective structuraliste est incapable de rendre compte de la possibilité qu'a un locuteur de produire et de comprendre une infinité de phrases. La créativité du locuteur démontre que la grammaire d'une langue dépasse le simple arrangement des unités dans un corpus fini. A partir de ces constations, Chomsky a fondé la grammaire générative et transformationnelle, qui considère le langage comme un processus avec un ensemble de règles et d'instructions dont l'application produit les énoncés admissibles de cette langue. Cette grammaire a comme pilier principal la syntaxe. En effet, les linguistes remarquent que le pouvoir créateur du locuteur se manifeste surtout dans le domaine de la syntaxe, par l'aptitude à former et à comprendre une infinité de phrases.

D.2. Les niveaux du langage

Le langage est un système de signes vocaux articulés qui permettent à l'homme de s'exprimer. Edelman suggère que les changements anatomiques de la trachée (vocal tract) au cours de l'évolution de l'homme, qui figurent parmi les caractéristiques permettant de distinguer les hommes des autres primates, sont eux-mêmes la racine du langage. En effet, ces changements de la trachée créent la structure articulatoire du langage qui mène à l'extension des capacités vocales et à la réceptivité de l'enfant aux modèles de sons (**[Vihman96]**).

Le langage peut être étudié à plusieurs niveaux. Dans notre travail, nous abordons les différents niveaux en nous attardant principalement aux niveaux consacrés à la syntaxe et à la sémantique. En effet, notre application (ou les perspectives de celle-ci) opéreront essentiellement à ces deux niveaux. Reprenons donc les différents niveaux du langage donnés par **[Deville98]** :

- La ***phonétique*** s'occupe de la description et de la production de sons possibles.
- La ***phonologie*** s'intéresse à l'inventaire de sons utilisés pour convoyer l'information dans un certain langage.
- La ***morphologie*** étudie la composition et la dérivation des mots.
- La ***syntaxe*** essaie de sectionner la séquence de mots d'un langage avec l'aide d'une grammaire.
- La ***sémantique*** s'intéresse à une manière d'utilisation des structures syntaxiques pour dériver la signification de la séquence de sons initiale.
- La ***pragmatique*** est l'étude des principes qui déterminent le sens particulier d'un énoncé en rapport avec les données contextuelles (autres énoncés, connaissance encyclopédique, intonation du locuteur).

Evoquons rapidement chacun de ces niveaux sans vouloir ré-écrire **[de Saussure 71]** et les nombreux ouvrages consacrés à ce sujet.

a) Phonétique

La phonétique étudie la nature physique et physiologique des distinctions de sons. La majorité des sons du langage sont le fait du passage d'une colonne d'air venant des poumons, qui traverse un ou plusieurs résonateurs de l'appareil phonatoire (le pharynx, la cavité buccale, la cavité labiale et les fosses nasales) (**[Yule96a]**).

La présence ou l'absence d'obstacles sur le parcours de la colonne d'air modifie la nature du son produit. Selon la forme et le volume du conduit vocal, modifié par les organes mobiles (lèvres, langue, voile du palais, luette), on obtient des sons différents (**[Napoli96]**). La distinction entre voyelles et consonnes s'effectue de la manière suivante :

- Si le passage de l'air se fait librement à partir de la glotte, on a affaire à une ***voyelle***.
- Si le passage de l'air à partir de la glotte est obstrué, complètement ou partiellement, en un ou plusieurs endroits, on a affaire à une ***consonne***.

L'articulation des sons est particulièrement importante pour le classement des consonnes et dépendante de trois caractéristiques dans les organes articulatoires :

- ***L'état des cordes vocales*** : la vibration ou non de celles-ci. La réalisation du son est dite ***sourde*** lorsque les cordes vocales ne vibrent pas ; si celles-ci entrent en vibration, la réalisation sera dite ***sonore***.
- ***Le mode d'articulation*** : la manière dont l'air est modifié en passant la trachée ou la glotte.
- ***Le lieu de l'articulation*** ou ***point d'articulation*** : l'endroit où se trouve l'obstacle au passage de l'air.

Tandis que les consonnes sont articulées via la fermeture ou l'obstruction de la trachée, les voyelles sont articulées sans obstacles au passage de l'air. Le seul traitement que l'air peut dès lors subir est la résonance (c'est-à-dire le renforcement de certaines bandes de fréquences).

Nous pourrions écrire ces sons en nous servant de notre l'alphabet. Cependant, il ne faut pas oublier que les lettres de l'alphabet romain n'ont pas nécessairement la même prononciation. Ex. : la lettre « e » n'a pas la même prononciation en anglais ou en français. C'est pourquoi il a été utile de produire un alphabet, séparé avec des symboles qui représentent les sons, appelé *alphabet phonétique*.

b) Phonologie

La phonologie étudie la distribution et l'agencement des sons du langage selon leur fonction dans le système de communication linguistique. Comment la phonologie se distingue-t-elle de la phonétique ? La phonétique et la phonologie ont le même objet d'étude, mais elles l'abordent sous des angles entièrement différents. La phonétique s'intéresse aux caractères physiques et physiologiques (articulatoires) du son ; la phonologie s'intéresse à la fonction linguistique du son, à sa capacité de porter une information. Chaque individu prononcera le mot « me » d'une manière physiquement différente car l'anatomie est différente, si l'individu souffre d'un mal de gorge ou s'il est énervé, ... Donc, « me » peut être prononcé comme [mi], [ni], [si]. Par cet exemple, nous avons bien trois unités phonétiques distinctes qui renvoient exactement au même sens et apportent la même information grâce à la phonologie. L'unité de base avec laquelle opère la phonologie est le *phonème* (**[Yule96b]**).

Selon **[Yule96b]**, nous pouvons constater que si nous substituons un son à un autre dans un mot et qu'il y a un changement de sens, les deux sons représentent des phonèmes différents. Cette propriété est le test opérationnel de base pour déterminer les phonèmes qui existent dans un langage. De cette manière, /f/ et /v/ sont deux des phonèmes de l'anglais car nous pouvons voir qu'il y a contraste de sens entre les formes *fat* et *vat*, *fine* et *vine*.

c) Morphologie

La segmentation du flux de paroles à un niveau phonologique est différent du niveau morphologique. Le niveau phonologique ne donne pas d'informations sur la façon dont le mot a été formé.

La morphologie qui signifie « l'étude de formes », décrit la structure interne des mots, comment ils sont composés, comment ils sont dérivés pour former différentes classes de mots et formes grammaticales ([**Deville98**]). Les éléments mis en évidence par cette étude sont connus sous le nom de morphèmes (l'unité minimale de signification ou une fonction grammaticale qui peut être retirée d'un mot). Prenons un exemple tiré de [**Yule96c**] : le mot « tourists ». Nous pouvons constater qu'il y a trois morphèmes :

- tour : unité significative.
- -ist : unité signifiant une personne qui fait quelque chose.
- -s : unité indiquant le pluriel.

d) Syntaxe

La syntaxe est l'étude des principes qui régissent la distribution et l'agencement des morphèmes lexicaux pour former des phrases.

(1) Grammaire

Jusque maintenant, nous avons décrit les expressions linguistiques comme des séquences de sons qui peuvent être représentées phonétiquement ou morphologiquement. Cependant, nous n'avons pas tenu compte du fait que ces mots peuvent seulement être combinés dans un nombre limité de modèles. Donc, nous avons besoin d'une manière de décrire la structure des groupes de mots et des phrases et c'est là qu'intervient la notion de grammaire ([**Charniak93d**]).

Une grammaire est constituée de :

- Un ensemble de ***symboles terminaux*** (mots et ponctuation du langage) qui apparaissent dans la phrase donnée par une personne.
- Un ensemble de ***symboles non-terminaux*** qui sont nécessaires à la description du langage mais ne figurent pas dans la phrase donnée par une personne.
- Un ***symbole non-terminal*** spécifique désigné comme le symbole de départ.

- Un ensemble de ***règles de réécriture*** qui transforment un symbole non-terminal en symboles terminaux et/ou symboles non-terminaux. Les différents types de règles sont vus au paragraphe 2 ci-dessous.

Par symboles non-terminaux, **[Charniak93c]** signifie les symboles qui sont dérivés en d'autres chaînes de symboles. Nous pouvons citer :

- ***S*** : désignant les phrases
- ***Np*** : désignant les syntagmes nominaux
- ***Vp*** : désignant les syntagmes verbaux
- ***Pp*** : désignant la préposition
- ***Noun*** : désignant les substantifs
- ***Verbe*** : désignant les verbes
- ***Det*** : désignant les déterminants
- ***Fpunc*** : désignant les caractères de ponctuation

Des terminaux repris dans **[Yule96d]** ont été répertoriés comme suit :

- ***Noms***: mots utilisés pour référer à des personnes, des objets, des créatures, des places, des idées abstraites, ...
- ***Adjectifs***: mots généralement utilisés avec des noms pour fournir plus d'informations (*happy* people, *large* objects, *stupid* ideas).
- ***Verbes***: mots utilisés pour faire référence à différents types d'action (run, jump) et d'états (be, seem).
- ***Adverbes***: mots utilisés pour fournir plus d'informations pour les actions mais ils peuvent également accompagner des adjectifs (*really* large, *very* stupid).
- ***Prépositions*** : mots utilisés avec des noms pour indiquer le temps (*at* five, *in* the morning), un lieu (*on* the table, *near* the window) et d'autres connections (*with* a knife, *without* a thought).
- ***Pronoms*** : mots utilisés à la place de noms pour référer à des choses déjà connues (*he* likes *himself*, *this* is *it*).
- ***Conjonctions*** : mots utilisés pour indiquer des relations entre des évènements ou des choses (we swam *although* it was very cold).

Une grammaire permet de générer toutes les structures correctes du langage et de détecter les phrases qui ne répondent pas à la structure du langage. Une phrase appartient au langage s'il existe une dérivation pour cette phrase : à partir du symbole S (symbole initial), on applique la séquence de règles jusqu'à ce que la chaîne obtenue ne contienne plus aucun symbole non-terminal. Nous pouvons distinguer deux types de dérivations :

- Dérivation la plus à ***gauche*** : dérivation dans laquelle à chaque pas on réécrit le symbole non terminal le plus à gauche.
- Dérivation la plus à ***droite*** : dérivation dans laquelle à chaque pas on réécrit le symbole non terminal le plus à droite.

Les grammaires ont la particularité qu'il est possible de représenter leurs dérivations de manière arborescente. L'arbre de dérivation (**figure 1**) est construit à partir du symbole initial (S) qui sera la racine et les feuilles représenteront l'ensemble des terminaux. Pour la réécriture d'un symbole non-terminal, nous devrons attacher au noeud correspondant à ce symbole autant de sous arbres que de symboles dans la partie droite de la règle utilisée (**[Deville98]**).

D'un point de vue pratique, étant donné une grammaire, nous allons décrire l'arbre de dérivation associé à celle-ci lors de l'analyse d'une certaine phrase. Prenons la grammaire suivante :

S-dep → S Fpunc
S → Np Vp
Vp → Verbe
Vp → Verbe Np
Vp → Verbe Np Np
Np → Det Noun Noun
Np → Noun
Det → the
Noun → dog
Noun → salespeople
Noun → biscuits
Verb → ate
Verb → sold
Fpunc → .

Si nous analysons la phrase : « *the dog ate.* » avec notre grammaire (dérivation à gauche), nous pouvons dégager l'arbre de dérivation suivant :

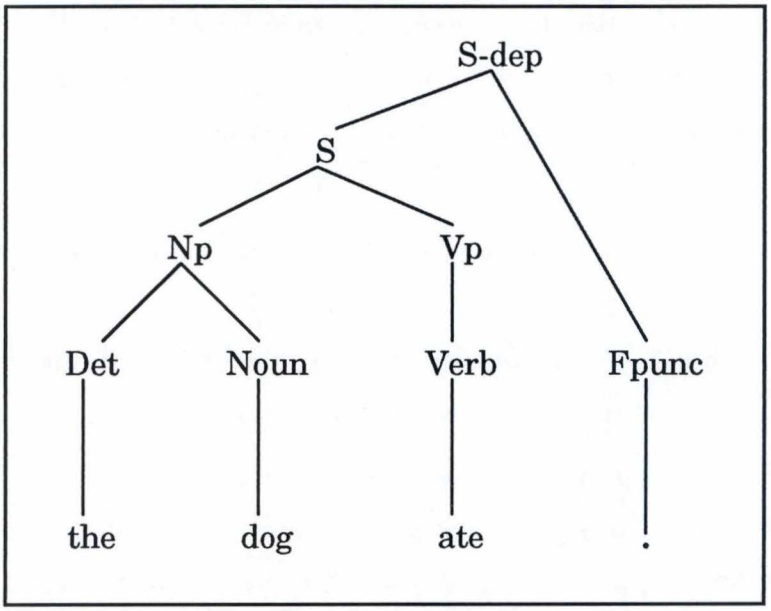


Figure 2 Arbre de dérivation

(2) Méthodes de parsing

Bien qu'il existe un grand nombre de grammaire, **[Deville98]** fait ressortir les types principaux suivants :

- ***Grammaire de type 0*** : il n'y a pas de restrictions, au moins un règle ne satisfait pas aux 3 conditions ci-dessous.
- ***Grammaire de type 1*** (dépendante de contexte) : il ne peut exister de règles $\alpha \rightarrow \beta$ où $|\alpha| \leq |\beta|$. Pas de règles raccourcissantes.
- ***Grammaire de type 2*** (indépendante de contexte) : pour toutes les règles $\alpha \rightarrow \beta$, la partie gauche doit contenir un non-terminal et la partie droite un chaîne non-nulle de symboles.
- ***Grammaire de type 3*** (régulière) : pour toutes les règles $\alpha \rightarrow \beta$, la partie gauche doit contenir un non-terminal et la partie droite doit contenir :
  - ◊ un terminal unique
  - ◊ un terminal et un non-terminal

Pour déterminer si une phrase appartient au langage, nous utiliserons un automate. Un automate est un mécanisme doté d'une tête de lecture et d'un ensemble d'états. A chaque instant de la reconnaissance,

l'automate se trouve dans un état donné. Il cherche à parcourir la phrase soumise en déplaçant la tête de lecture vers la droite sur base d'un ensemble de fonctions de transitions (règles). C'est ce qu'on appelle une procédure de reconnaissance. Un réseau, quant à lui, est la représentation graphique de l'ensemble des mouvements licites d'un automate (**[Deville98]**).

Nous venons de définir un « reconnaisseur ». Il nous faut également définir ce qu'est un analyseur syntaxique. **[Deville98]** définit un analyseur syntaxique comme un reconnaisseur qui prend note de l'histoire dérivationnelle de la phrase, détermine si une phrase appartient au langage, et si oui, retourne l'ensemble des structures ou arbres de dérivation (**figure 1**) qui peuvent être associées à cette phrase. On distingue deux types d'orientation à un analyseur syntaxique :

- ***Descendante*** : départ de l'analyse à partir du symbole initial de la grammaire et construction d'une représentation de la phrase de haut en bas.
- ***Ascendante*** : départ de l'analyse à partir des mots de la phrase (terminaux de la grammaire) et réduction de ces constituants en constituants de niveau supérieur jusqu'à l'obtention du symbole initial de la grammaire.

Les grammaires vont nous être particulièrement utiles pour le développement futur de notre application. En effet, si nous extrayons les éléments syntaxiques d'une phrase, nous sommes en mesure de repérer les éléments nécessaires à une certaine compréhension de cette phrase. Une fois ces éléments disponibles, il nous restera à formuler une réponse en fonction de la nature de la phrase que nous aurons relevé précédemment.

e) Sémantique

La sémantique est l'étude de la signification de mots, de phrases ou de textes. Nous allons présenter différentes informations qui peuvent nous être utiles pour extraire la signification (d'un mot, d'une phrase ou d'un texte) à savoir : les caractéristiques sémantiques, les rôles sémantiques et les relations sémantiques.

(1) Caractéristiques sémantiques

Si nous prenons la phrase « The hamburger ate the man on the table», nous pouvons voir que celle-ci est syntaxiquement correcte mais est vraiment étrange d'un point de vue sémantique. Trouvons pourquoi.

Comme on le constate dans **[Yule96e]**, nous pouvons expliquer cela par le fait que les caractéristiques sémantiques du mot « hamburger » sont différentes de celles du mot « man » lorsque ces mots sont utilisés avec le verbe « ate ». En fonction de cette observation, essayons d'être plus général. Nous allons déterminer le composant crucial qu'un nom doit avoir pour être sujet du verbe « ate ». Un tel composant doit correspondre à un « être animé ». Donc, les caractéristiques sémantiques deviennent *+animate* (pour un être animé) et *-animate* (pour une chose). A partir de là, nous pouvons construire des tableaux de caractéristiques sémantiques toujours plus complexes. Pour donner un exemple, établissons les caractéristiques sémantiques des mots de notre phrase introductive « The hamburger ate the man on the table », ce qui donne sous forme de tableau :

car. Sém./mot	**hamburger**	**man**	**table**
animate	-	+	-
human	-	+	-
male	-	+	-
adult	-	+	-

Tableau 1 Caractéristiques sémantiques de mots

Cependant, dans certains cas, il est difficile de trouver des caractéristiques pour certains mots du langage. Ex. : advice, warning, ... On qualifiera alors ces mots de « conteneurs » transportant des composantes de signification.

(2) Rôles sémantiques

Au lieu de penser les mots comme « conteneurs », nous pouvons regarder leurs rôles dans la situation décrite par la phrase. Ex. : « The boy kicked the ball » où le verbe décrit une action.

Nous pouvons identifier quelques rôles sémantiques repris dans **[Yule96e]** :

- ***Agent*** : entité qui produit l'action. Ex. : *the boy*.
- ***Thème*** : entité qui est impliquée ou affectée par l'action. Ex. : *the ball*.
- ***Instrument*** : agent qui utilise une autre entité pour produire une action. Ex. : *eating with a spoon, writing with a pen*.
- ***Expérimentateur*** : lorsqu'une phrase désigne une entité comme une personne qui a un sentiment, une perception de quelque chose ou un état. Ex. : « Did you hear that noise? », *you* est expérimentateur.
- ***Location*** : indique où se trouve l'entité. Ex. : *on the table*, *in the room*.
- ***Source*** : location à partir de laquelle une entité bouge. Ex. : *from savings* to checking.
- ***But*** : location vers laquelle l'entité se déplace. Ex. : from savings *to checking*.

(3) Relations sémantiques

Les mots peuvent aussi avoir des relations. Ex. : la signification de « shallow » est l'opposé de la signification de « deep », ...

[Yule96e] et **[Miller et al. 93]** ont précisé les différentes relations lexicales que nous pouvions rencontrer :

- ***Synonyme*** : mots de même catégorie avec des significations très proches. Ex. : *answer* et *reply*, *liberty* et *freedom*.
- ***Antonyme*** : mots qui ont des sens contraires. Ex. : *big* et *small*, *male* et *female*.
- ***Hyponyme*** : la signification d'un mot est inclue dans la signification d'un autre mot. Ex. : *dog* et *animal*, *carrot* et *vegetable*. Nous pouvons aussi dire que « horse » et « dog » , tous deux hyponymes de « animal », sont *co-hyponymes* de « animal ».
- ***Homophone*** : mots d'orthographes différentes qui ont la même prononciation. Ex. : *bear* et *bare*, *meat* et *meet*.

- ***Homonyme*** : mot de même prononciation qu'un autre mais d'orthographe et de signification différentes ou de même orthographe mais de signification différente. Ex. : *bank* (digue) et *bank* (banque), *pupil* (élève) et *pupil* (pupille).
- ***Polysème*** : mot qui présente plusieurs significations. Ex. : *foot* (d'une personne, d'un lit, d'une montagne).
- ***Métonyme*** : relation entre des mots basée sur une connection entre ces mots. Ex. : *car* et *wheels* (relation « part de »), *bottle* et *coke* (relation de conteneur-contenu), *king* et *crown* (relation sociale, de travail).
- ***Collocation*** : organisation de mots qui se produisent souvent ensemble. Ex. : *butter* et *bread*, *salt* et *pepper*.

Le chapitre III.B. expose WordNet, qui est un outil développé par les chercheurs en intelligence artificielle et en psycholinguistique. Cet outil recense un grand nombre de mots anglais et les relations sémantiques qu'ils ont entre eux. Ces relations nous ont été utiles lorsque nous avons procédé à des recherche de synonymes et d'hypernymes dans l'application réalisée. Ces fonctions sont expliquées en détails dans le chapitre III.

f) Pragmatique

La pragmatique étudie comment le langage est utilisé dans un contexte social. Elle précise comment les phrases sont construites dans le flux de la conversation, comment notre pensée est formulée en paroles et comment les degrés de formalité et de politesse sont signalés (**[Yule96e]**).

On parle souvent de « signification invisible » car nous reconnaissons ce que quelque chose veut dire sans que cela ait été dit ou écrit.

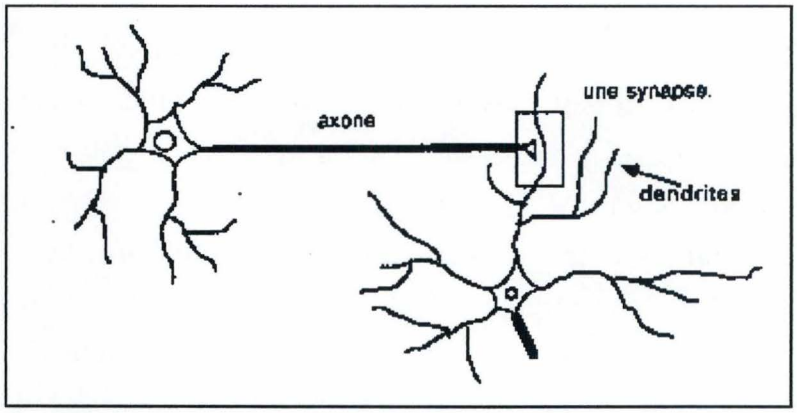
E. Les neurosciences[1]

E.1. Introduction

Les neurosciences étudient le fonctionnement du cerveau humain. L'apport de cette discipline est évident pour les sciences cognitives. Les chercheurs en intelligence artificielle essayent de construire de manière précise des modèles de réseaux. Dans ce bref exposé, nous nous limitons à définir ce qu'est un neurone, et d'expliquer certains phénomènes qui se produisent entre ces neurones. Nous tentons aussi l'explicitation d'une hypothèse relative à l'apprentissage.

E.2. Neurone

Le cerveau est séparé en plusieurs parties bien distinctes. On suppose qu'une partie est réservée à des fonctionnalités du langage. Cette hypothèse est renforcée par le fait que lors d'accidents ayant endommagé seulement une partie du cerveau, des patients ont des troubles du langage. Le cerveau est constitué d'un grand nombre de neurones (10^{11}) reliés entre eux et au système nerveux central (CNS - Central Nervous System). Il existe différents types de neurones, mais leur structure de base est généralement identique.



Un neurone est partagé en trois parties bien distinctes : le corps de la cellule, les dendrites et l'axone. Les dendrites sont semblables à des « petits tubes » et c'est par elles que les signaux entrent dans le neurone. L'axone permet de véhiculer l'information d'une cellule vers une autre partie du

[1] *Source : **[Powers et al. 89b]***

cerveau. Un neurone typique possède de 1000 à 10000 synapses, et peut recevoir de l'information de 1000 autres neurones. Chaque neurone peut « tirer », c'est à dire émettre un signal. La possibilité d'émission du signal dépend du nombre de dendrites qui sont activées. Certaines dendrites sont inhibitrices et réduisent la possibilité que le neurone associé tire, ou au contraire excitent et augmentent la possibilité que le neurone associé tire. En fait, chaque dendrite possède un certain « poids », et il faut que la somme de tous les poids excède un certain seuil pour que le neurone tire.

Lorsqu'un neurone vient de tirer, pendant une courte période suivant cette action, il n'est plus possible pour la cellule de tirer. Ensuite, le seuil revient progressivement vers sa valeur de base. Un neurone ne peut donc pas être tiré sans arrêt. Il existe une *fréquence de tir*.

Les scientifiques ont également constaté que les neurones sont organisés en couches et que les neurones d'une couche particulière ont des destinations particulières dans le cerveau.

Il semble aussi que l'activité qui prend place dans le cerveau est largement locale, c'est à dire que ce sont les cellules très proches qui travaillent ensemble (1 ou 2 mm^2).

E.3. Modélisation d'un réseau de neurones

Le but des chercheurs des sciences cognitives est bien sûr de modéliser le plus fidèlement possible des réseaux neuronaux. Il faut toutefois être attentif à certains problèmes pouvant survenir lors de la modélisation, étant donné que l'on connaît mal le fonctionnement exact du cerveau. Une question cruciale est bien sûr de se demander comment les neurones peuvent « apprendre ». Différentes hypothèses existent, dont une soutenant le fait qu'une synapse ayant contribué à faire tirer le neurone auquel elle apporte l'information (neurone post-synaptique) va voir son poids renforcé (*plasticité*). Si on n'y prend pas garde, une telle théorie peut conduire le réseau à la *saturation*. Différentes hypothèses existent aussi sur la manière d'éviter ce problème. L'une d'elle est le fait de déclarer que le poids de l'entièreté des synapses doit rester constant. Un exemple d'une

telle approche est celle du réseau à organisation automatique[2] de Malsburg **[Malsburg73]**.

Le chapitre II.C. est consacré exclusivement à la modélisation des réseaux neuronaux.

F. Les questions de l'apprentissage

F.1. Introduction

Les scientifiques (cognitifs) se sont évidemment intéressés aux questions de l'apprentissage. Bien qu'aucune théorie complètement satisfaisante n'ait vu le jour, certains mécanismes régissant notre capacité à apprendre semblent évidents. **[Powerset al. 89c]** affirme que les hommes, consciemment et inconsciemment, font des hypothèses, construisent des modèles, et révisent ces derniers selon leurs expériences. On peut alors attirer l'attention sur certains mécanismes de raisonnement, qui aident à mieux comprendre la tâche qu'est l'apprentissage. Dans ce chapitre, nous expliquons brièvement les mécanismes généraux de l'apprentissage, ainsi qu'un facteur extérieur (l'erreur) susceptible d'influencer grandement ceux-ci. Nous nous tournons alors vers le domaine particulier de l'apprentissage qu'est l'apprentissage du langage naturel et introduisons différentes techniques utilisées en intelligence artificielle pour modéliser l'acquisition du langage.

F.2. Différents types d'apprentissage

Le *raisonnement par déduction* est en fait une transformation du plus général vers le plus spécifique (on part de règles ou de principes pour arriver à des instances ou des cas). La *spécialisation* est un processus déductif qui instancie une partie ou la totalité des variables d'une proposition pour obtenir une instance de cette proposition. C'est ce processus qui est impliqué quand une grammaire est utilisée pour la reconnaissance ou la génération du langage. La *spécification* est un processus complémentaire au raisonnement déductif et permet de restreindre l'utilisation de règles générales. Ce processus serait actif lors de

---

[2] *self-organizing network*

l'utilisation des grammaires génératives et transformationnelles (voir chapitre I.D.). Ce processus de raisonnement par déduction est principalement utilisé pour traiter des connaissances déjà stockées. Il existe un autre processus, qui semble être beaucoup plus fondamental au phénomène de l'apprentissage : le *processus inductif*.

Le processus inductif utilise les cas particuliers pour arriver à des règles plus générales. Un ensemble de cas particuliers peut faire ressortir des similarités, permettant de suggérer un ensemble de règles capables de caractériser ces cas, de les générer, et de tester leur appartenance à l'ensemble.

On constate que les hommes n'ont pas de difficultés importantes pour décrire des ensembles d'objets à partir de règles générales. Par contre, il semble que les processus déductifs autres que l'instanciation posent plus de problèmes (pour accepter les résultats, expliquer le processus utilisé...). Des processus inductifs sont couramment utilisés par les individus, mais ces processus sont très difficiles à expliquer formellement. Pour cette raison, les auteurs pensent que les hommes possèdent un processus d'induction inconscient. Un phénomène courant du processus d'induction est celui de la *sur-généralisation*, qui se produit lorsque les règles générales sont appliquées trop souvent. Ce processus semble d'ailleurs assez présent lors de l'apprentissage de la langue maternelle par les enfants **[Powers et al. 89d]**. Un des indices en faveur de l'existence d'un tel système vient de l'observation d'enfants apprenant la langue anglaise. Ces observations montrent que ceux-ci commencent par répéter les verbes qu'ils entendent, sans les comprendre, puis petit à petit semblent construire des règles régissant ces verbes (terminaison -ed au passé...). Les enfants ne tiennent à ce moment aucun compte des verbes irréguliers et appliquent les règles générales aveuglément. Ce n'est que plus tard qu'ils prennent la distinction en compte.

F.3. L'apprentissage du langage naturel

Un grand nombre de paramètres peuvent influencer l'acquisition du langage ou encore donner des indications sur la façon dont l'acquisition se réalise. **[Brown et al. 63]** sont en faveur du fait que les hommes posséderaient une grammaire interne :

« In the natural situation of the child with his family the best evidence that he posseses construction rules is the occurrence of systematic errors. So long as a child speaks correctly, it is possible that he says only what he has heard. In general we cannot eliminate the possibility of an exact model for each sentence that is put out. However when a small boys say « I digged in the yard », it is unlikely that he is imitating. Furthermore, his mistake is not a random one. We can see how he might have made it by overgeneralizing certain existent regularities »

Cette observation tend donc à prouver que les hommes possèdent une grammaire interne, et qu'elle est mise à jour lors de l'apprentissage du langage.

Un des facteurs influant sur l'apprentissage que nous avons rencontré le plus souvent dans nos lectures est l'erreur. Il existe encore beaucoup d'incertitudes quant au rôle exact de l'erreur. On peut par exemple se demander quelle influence ont les parents lorsqu'ils corrigent les erreurs de langage de leurs enfants. **[Bloom93]** donne un aperçu de différentes hypothèses propres à l'acquisition du langage naturel, et relate de nombreuses observations et expériences réalisées principalement sur des enfants.

Ces observations peuvent directement être intégrées dans le développement de modèles par les chercheurs en intelligence artificielle. Par exemple, ils peuvent concevoir un programme susceptible d'apprendre le langage naturel, mais en prévoyant un système pour que des personnes extérieures supervisent cet apprentissage, et éventuellement corrigent des erreurs, ou encore donnent des exemples. **[Brill] [Brill92] [Brill93]** a présenté différents travaux relatifs au traitement du langage naturel dans lesquels il prévoit une supervision humaine.

F.4. La modélisation du langage

Pour mettre en pratique une méthode d'apprentissage du langage, il est nécessaire que celui-ci soit représenté d'une certaine manière. Cette modélisation comporte bien sûr de nombreuses possibilités. Ainsi, nous pouvons représenter la structure du langage sous forme de grammaire (voir

chapitre I.D). Celui-ci peut également être représenté d'une manière plus mathématique sous formes de chaînes de Markov (voir chapitre II.A.). Bien sûr, d'autres méthodes que nous n'avons pas abordées dans ce mémoire existent encore.

F.5. La modélisation des processus d'apprentissage du langage

Comme nous l'avons indiqué dans le chapitre I.C, il existe une distinction entre l'intelligence artificielle forte et faible, selon que l'on essaye de modéliser le fonctionnement réel du système d'apprentissage présent chez les hommes ou que l'on se contente d'en imiter les résultats. Il existe un très grand nombre de techniques permettant de modéliser avec plus ou moins de succès le processus d'apprentissage du langage.

Certaines techniques peuvent incorporer des hypothèses relatives à l'acquisition du langage. Ainsi, WordNet, que nous examinerons en détails dans le chapitre III.B., est basé sur l'hypothèse de l'existence de concepts (acquis ou innés). La méthode de classification (*clustering*), décrite dans le chapitre II.B. comporte l'hypothèse que le processus d'apprentissage classifie les mots selon certains critères **[Powers et al. 89d]**.

Les limitations du système cognitif sont aussi une bonne indication dans le développement de processus censés imiter ceux des hommes. Certaines d'entre elles sont reprises dans **[Powers et al. 89e]**.

Des techniques comme l'apprentissage statistique du langage ne s'appuient pas vraiment sur des bases psycholinguistiques mais essayent d'utiliser des techniques statistiques pour construire des grammaires et les utiliser par la suite. De telles techniques sont décrites par **[Charniak93]** et **[Samuelsson et al. 97]**.

Nous pensons qu'il n'est pas possible de déterminer un modèle du langage, qui soit exact dès le début de son utilisation. Il est nécessaire que tout modèle soit raffiné, expérimenté et modifié sans cesse pour approcher le comportement des hommes dans l'utilisation du langage naturel.

Notre travail se situe sans aucun doute dans l'intelligence artificielle faible. En effet, même si nous pouvons utiliser des techniques dérivées de suppositions psycholinguistiques, le but de celui-ci n'est en aucun cas de

comprendre le fonctionnement réel du processus d'apprentissage du langage ou la façon dont celui-ci est représenté dans le cerveau de hommes. Le but du programme réalisé est d'**imiter** le mieux possible le comportement de personnes qui dialoguent à propos des sujets divers.

Ce que nous appelons « apprentissage du langage » dans ce contexte, est toute méthode qui permettrait à l'ordinateur d'acquérir ou d'affiner un modèle du langage. Ainsi des méthodes permettant de générer une grammaire, ou encore d'enregistrer des erreurs fréquentes commises peuvent être vues comme des méthodes d'apprentissage du langage.

Dans les chapitres suivants, nous introduisons différentes techniques d'apprentissage, certaines issues directement des recherches en sciences cognitives (modélisation des réseaux neuronaux), d'autres purement statistiques (apprentissage statistique). Pour chacune de ces techniques, nous voyons en quoi elle peut être utile dans l'apprentissage du langage.

Nous verrons dans le chapitre V. la manière dont les techniques étudiées (apprentissage statistique, classification et réseaux neuronaux) peuvent nous servir dans les développements futurs de l'application que nous avons implémentée.

Chapitre II
L'apprentissage par les machines

A. L'apprentissage statistique du langage

A.1. Introduction

L'apprentissage statistique du langage est fortement utilisé dans la pratique. Un des défis de cette discipline est de tenter d'établir un modèle, le plus réaliste possible, du langage naturel. Pour cela, **[Charniak93]** décrit un modèle du langage basé sur le Modèle de Markov Caché (*HMM*). Nous intégrons ce chapitre dans notre mémoire car nous considérons que l'intégration d'un module d'apprentissage du langage constitue une étape importante dans l'évolution de notre programme[1]. Dans ce chapitre, nous commençons par indiquer un modèle du langage (modèle du trigramme). Nous voyons alors que ce modèle du langage peut être vu comme une chaîne de Markov (cachée), que nous définissons. Ensuite, nous expliquons plusieurs algorithmes très souvent utilisés pour gérer les chaînes de Markov (cachées), et qui nous aideront à manipuler notre représentation du langage. Pour terminer ce chapitre, nous voyons alors de quelle façon nous pourrions nous servir de ce modèle en traitement du langage naturel.

---

[1] *Voir chapitre V.*

A.2. Le modèle n-gramme du langage

Le modèle n-gramme est un modèle du langage assez peu sophistiqué mais ayant beaucoup de succès auprès des chercheurs. Ce succès vient probablement de la simplicité du modèle, et de la disponibilité de logiciels intégrant celui-ci. Généralement, le n choisi est 3, d'où le nom *trigramme.*

L'hypothèse drastique suivie par ce modèle est de supposer que seuls les 2 mots précédents un mot influencent le choix (probabilité) de ce dernier. D'une manière formelle, nous pouvons exprimer ce fait comme suit :

Equation A-1

$$P(w_n|w_1 \cdots w_{n-1}) = P(w_n|w_{n-2}, w_{n-1})$$

Notre modèle statistique du langage est alors assez simple :

Equation A-2

$$\begin{aligned} P(w_{1,n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1,2}) \cdots P(w_n|w_{1,n-1}) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_{1,2}) \cdots P(w_n|w_{n-2,n-1}) \\ &= P(w_1)P(w_2|w_1)\prod_{i=3}^{n} P(w_i|w_{i-2,i-1}) \end{aligned}$$

où $P(w_n)$ représente la probabilité d'avoir le mot w_n, $P(w_n|w_1,\ldots,w_{n-1})$ est la probabilité d'avoir le mot w_n s'il est précédé des mots $w_1,\ldots w_{n-1}$ et $P(w_{1,n})$ est la probabilité d'avoir la séquence de mots $w_1 \ldots w_n$.

En supposant le fait qu'il existe deux « pseudo-mots » w_{-1} et w_0 qui seraient des indicateurs de début de texte, on peut écrire :

Equation A-3

$$P(w_{1,n}) = \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1})$$

ce qui permet de donner une forme plus élégante à l'équation.

A.3. Le modèle de Markov

Le concept de chaîne de Markov étant très proche de celui d'automate à états finis, nous commençons par définir formellement un automate à états finis :

- AF=(E,V,t,e_0,F) où
- E est un ensemble fini d'états
- V est le vocabulaire du langage
- t est une fonction de transition qui assigne un état à tous les couples de valeur (e,x) où e ∈ E et x ∈ V
- e_0 est l'état initial
- F est un sous-ensemble d'états de E, appelés états finals.

On pourrait représenter un « mini-langage » par un automate à états finis. Par exemple, un alphabet d'un langage pourrait n'avoir que deux symboles : *a* et *b*, et ne former que des mots se terminant par b. L'automate acceptant ce langage est représenté ci-dessous :

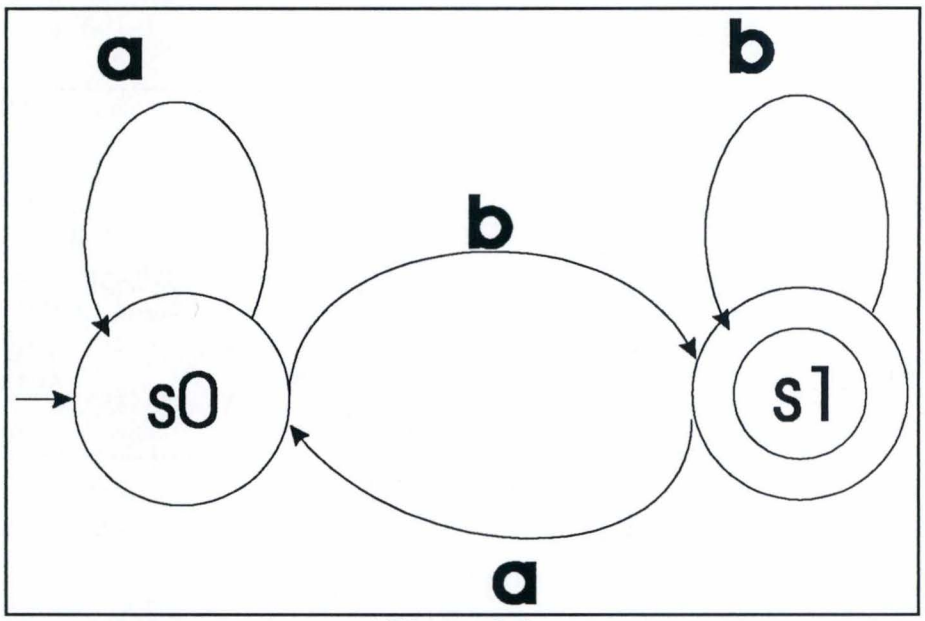


Figure A-1

On remarque qu'un tel automate peut être soit *accepteur* (il détermine si une chaîne fait partie du langage) ou *générateur* (il génère des membres du langage qu'il décrit). L'automate décrit dans la **Figure A.1** n'est pas déterministe, c'est à dire que lorsqu'on est dans un état, on ne peut pas toujours prédire l'état suivant. Or si on utilise notre automate comme un générateur, comment déterminer quelle transition appliquer? Il faut que les messages générés par notre automate (le modèle du langage) reflètent le mieux possible les messages générés par le langage modélisé. Un moyen d'approcher le langage réel est d'attribuer des probabilités à chacune des

transitions. On obtient alors une *chaîne de Markov*. Comme les automates, les chaînes de Markov peuvent être *acceptrices* ou *génératrices*. La somme des probabilités des arcs partant d'une transition doit être égale à un. On pourrait également appeler une chaîne de Markov un automate à états finis probabiliste.

Un point important à signaler est le fait qu'on peut travailler à plusieurs niveaux du langage[2] avec les chaînes de Markov, suivant l'utilisation qu'on veut faire du modèle. Ainsi, si on veut modéliser la structure des mots anglais, dans le but de corriger l'orthographe par exemple, on va utiliser des trigrammes de lettres. On peut tout aussi bien utiliser des trigrammes de mots pour modéliser les séquences de mots dans le langage Anglais.

Décrivons un exemple de modélisation de trigrammes (de lettres) par une chaîne de Markov :

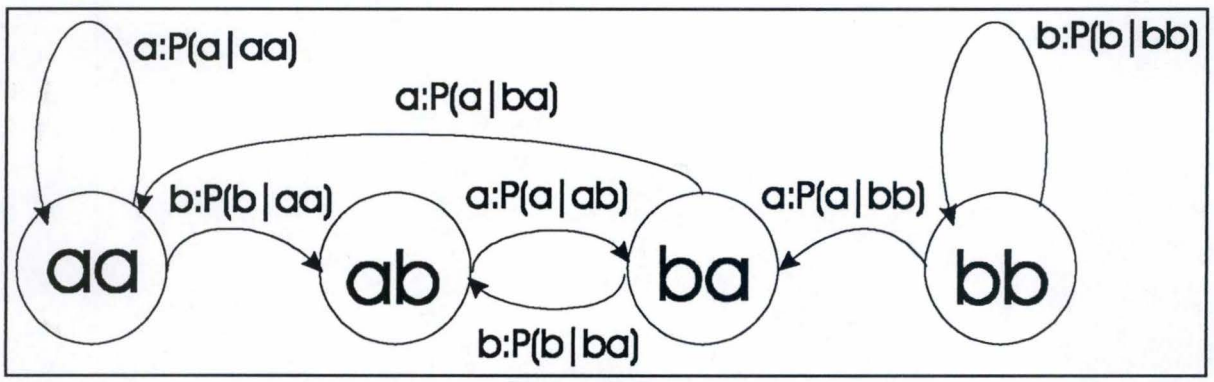


Figure A-2

Chaque état représente une séquence de deux lettres (contexte) et chaque transition est définie comme suit :

- Un état de départ représente le contexte
- Un symbole de sortie représentant la lettre suivant le contexte
- Une probabilité d'avoir un tel symbole de sortie
- Un état d'arrivée représentant le contexte pour le symbole de sortie suivant

Dans le « mini-langage » modélisé ci-dessous, si nous avons la séquence de lettres *ab* dans un mot, alors la probabilité que la lettre

[2] *Voir chapitre II.D*

suivante soit a est $P(a|ab)$. Le nouveau contexte est alors ba puisque nous ne prenons en compte que deux lettres.

A.4. Modèle de Markov caché (HMM)

a) Introduction

Dans la pratique, le modèle de Markov a montré qu'il était très peu précis. En effet, Jelinek[3] a réalisé un rapport indiquant qu'après avoir collecté des informations statistiques d'un corpus de 1.500.000 mots, et ayant appliqué le modèle de trigrammes résultant à un texte de 300.000 mots, 25 % des trigrammes du second texte n'apparaissaient pas dans le premier. Une solution a ce problème est d'utiliser également les bigrammes et unigrammes du texte initial. On a alors :

Equation A-4

$$P(w_n|w_{n-2},w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2},w_{n-1})$$

λ_1, λ_2 et λ_3 sont des constantes non négatives telles que $\lambda_1 + \lambda_2 + \lambda_3 = 1$. La valeur des paramètres λ sont des valeurs *raisonnables*, pouvant être attribuées automatiquement par des algorithmes d'entraînement.

Voici un exemple de chaîne de Markov cachée :

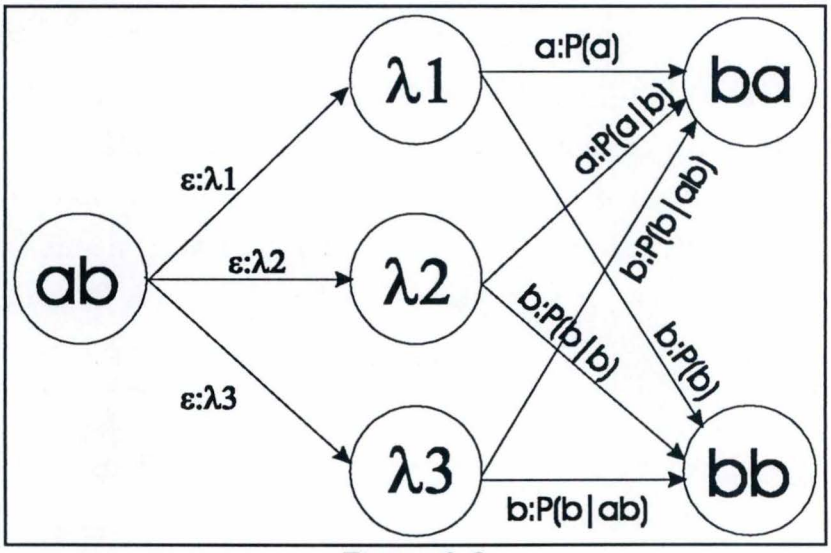


Figure A-3

[3] *Jelinek, F. Markov source modeling of text generation. In* The impact of processing techniques on communications, *J.K. Skwirzinski, Ed. Nijhoff, Dordrecht, 1985.*

Ce schéma fait clairement ressortir la caractéristique essentielle d'une chaîne de Markov cachée : ayant un ensemble de symboles de sortie, on ne peut indiquer de manière univoque la séquence d'états qui nous a permis de générer cette chaîne de sortie. Il existe des algorithmes (dont certains sont décrits dans ce chapitre) qui permettent de gérer efficacement ces chaînes de Markov cachées. Il est aussi possible de leur attribuer automatiquement les probabilités de leurs transitions. Nous pouvons maintenant définir le modèle de Markov plus formellement.

b) Modèle formel

On définit une chaîne de Markov cachée comme un quadruplet $<s^1,S,W,E>$ où S est un ensemble d'états, s^1 est l'état initial, W est un ensemble de symboles de sortie, et E un ensemble de transitions. Pour chacun des ensembles nous assumons un ordre des éléments :

- $<s^1,s^2,...s^\sigma>$
- $<w^1,w^2,...w^\omega>$
- $<e^1,e^2,...e^\varepsilon>$

Une transition est un quadruplet $<s^i,s^j,w^k,p>$ où $s^i \in S$ est l'état où la transition débute, $s^j \in S$ est l'état où la transition mène, $w^k \in W$ est un symbole de sortie soit généré soit accepté[4] et p est la probabilité que la transition soit choisie. Nous pouvons représenter une telle transition par $s^i \xrightarrow{w^k} s^j$. Il est à remarquer qu'entre deux états peut exister au plus une seule transition avec un même symbole de sortie. La probabilité p associée à une transition $s^i \xrightarrow{w^k} s^j$, c'est à dire $P(s^i \xrightarrow{w^k} s^j)$ est définie comme la probabilité qu'à un moment t la chaîne se trouve dans l'état s^i et émette le symbole w^k pour arriver au moment t+1 dans l'état s^j,. Notons la différence entre w^i qui représente le $i^{ème}$ symbole de sortie de l'ensemble W et w_i qui représente le caractère de sortie à la $i^{ème}$ unité de temps. On a donc

Equation A-5

$$P(s^i \xrightarrow{w^k} s^j) \overset{def}{=} P(S_{t+1} = s^j, W_t = w^k \mid S_t = s^i)$$

avec $t \geq 1$.

[4] *Suivant l'utilisation faite de la chaîne*

D'une manière abrégée, on écrira

Equation A-6

$$P(s^i \xrightarrow{w^k} s^j) = P(s^j, w^k | s^i)$$

L'utilisation des chaînes de Markov suppose que la probabilité d'un certain symbole en sortie, ou du prochain état est seulement influencée par l'état précédent (assomption de Markov). Nous avons alors :

Equation A-7

$$P(W_n, S_{n+1} | W_{1,n-1}, S_{1,n}) = P(W_n, S_{n+1} | S_n) = P(s^n \xrightarrow{w^n} s^{n+1})$$

Rappelons que la probabilité d'une séquence $w_{1,n}$ en sortie est égale à la probabilité de tous les chemins possibles à travers la chaîne de Markov cachée qui produit cette séquence. En d'autres mots on a

Equation A-8

$$P(W_{1,n}) = \sum_{S1,n+1} P(W_{1,n}, S_{1,n+1})$$

Nous pouvons calculer les probabilités d'une séquence de sortie particulière en utilisant notre HMM :

Equation A-9

$$\begin{aligned}
P(w_{1,n}) &= \sum_{S1,n+1} P(W_{1,n}, S_{1,n+1}) \\
&= \sum_{S1,n+1} P(S_1) P(W_1, S_2 | S_1) P(W_2, S_3 | W_1, S_{1,2}) \ldots P(W_n, S_{n+1} | W_{1,n-1}, S_{1,n}) \\
&= \sum_{S1,n+1} P(W_1, S_2 | S_1) P(W_2, S_3 | S_2) \ldots P(W_n, S_{n+1} | S_n) \\
&= \sum_{S1,n+1} \prod_{i=1}^{n} P(W_i, S_{i+1} | s_i i) \\
&= \sum_{S1,n+1} \prod_{i=1}^{n} P(S_i \xrightarrow{w_i} S_{i+1})
\end{aligned}$$

La seconde ligne de l'équation réécrit simplement la première ligne, en la développant. Ensuite, à la troisième ligne on simplifie l'équation précédente en tenant en compte que le premier état est toujours identique, et donc la probabilité de le voir apparaître est 1. De plus, on a appliqué l'assomption de Markov, qui stipule que seul l'état précédent de la chaîne

est important pour prévoir l'état suivant. Enfin, la dernière ligne réécrit la précédente d'une manière plus succincte. Il est clair que cette équation permettant de calculer la probabilité de sortie d'une HMM n'est pas très utile, puisqu'elle oblige à calculer tous les chemins possibles dans l'espace de recherche. Or, ce nombre de chemins possibles croît exponentiellement avec la longueur de la chaîne de sortie. Nous allons alors exposer maintenant des algorithmes plus efficaces permettant de travailler avec des chaînes de Markov cachées.

A.5. Algorithmes de traitement

a) Détermination de la séquence d'états la plus probable à partir des symboles de sortie (Viterbi)

Nous avons commencé par définir $\sigma(t)$ qui nous donne le chemin suivi à travers les états le plus probable étant donné une séquence de symboles de sortie particulière :

Equation A-10

$$\sigma(t) \overset{def}{=} \arg\max_{S_{1,t}} P(S_{1,t} | W_{1,t-1})$$
$$= \arg\max_{S_{1,t}} P(S_{1,t}, W_{1,t-1})$$

L'idée de base de l'algorithme est de calculer la séquence d'états la plus probable, en partant avec une séquence de sortie **vide**. On va alors calculer successivement les états par lesquels on est passé le plus probablement en fonction de chaque symbole de sortie émis.

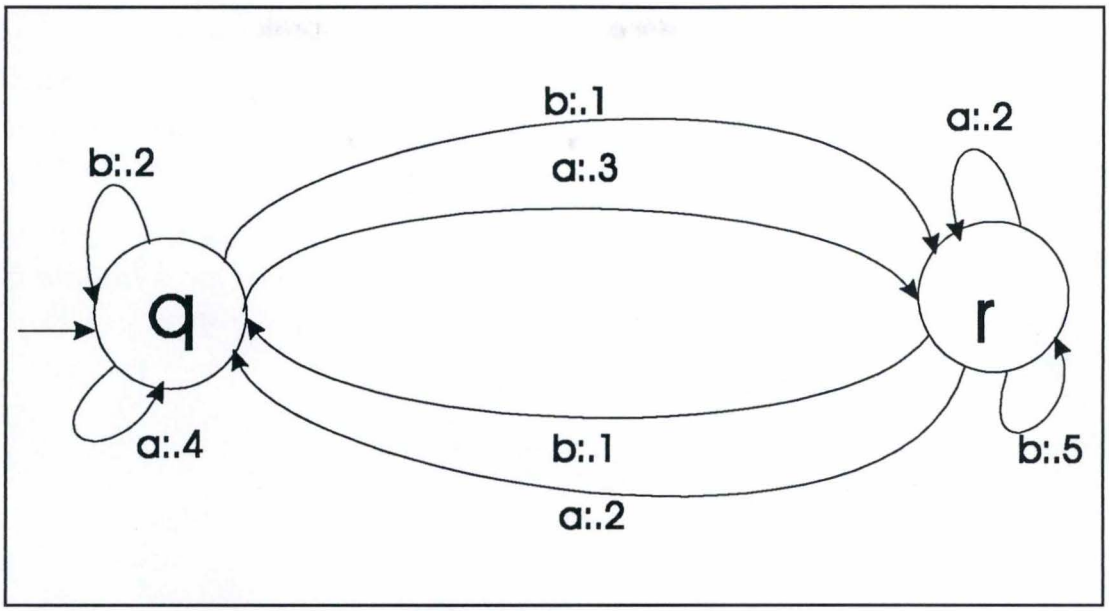


Figure A-4

Par exemple, on suppose qu'on a l'automate de la **figure A.4**. La séquence de sortie connue est « bbba ». On commence avec la séquence (de sortie) vide. A ce moment la probabilité d'être dans l'état q est 1 (puisque q est l'état initial). Ensuite nous considérons la séquence la plus probable après avoir obtenu le symbole « b » en sortie. Nous allons rester dans l'état q puisque la probabilité est de 0.2 (contre 0.1 pour nous amener dans l'état r). Pour le moment nous avons donc la séquence la plus probable qui est qq. Le symbole suivant est à nouveau un « b », nous allons donc toujours rester dans l'état q. On continue ainsi de suite, jusqu'à la fin de la chaîne de sortie. Dans ce cas, la séquence la plus probable d'états est « qqqqq ». Nous pouvons calculer la probabilité totale en multipliant chacune des probabilités employées. Ici nous avons donc 1*0.2*0.2*0.2*0.4=0.0032.

Maintenant écrivons la description formelle de l'algorithme :

Equation A-11

$$\sigma_i(t) \overset{def}{=} \arg\max_{S1,t} P(W_{1,t-1}, S_{1,t-1}, S_t = s^i)$$

On a aussi :

Equation A-12

$$\sigma_i(1) = s^i$$

$$\sigma_i(t+1) = \sigma_j(t) \circ s^i, j = \arg\max_{k=1}^{\sigma} P(\sigma_k(t)) P(s^k \xrightarrow{w_t} s^i)$$

Le symbole ° signifie « concaténation ». Cette équation établit le fait que, connaissant le meilleur chemin jusqu'au temps t, alors la meilleure façon de déterminer l'état au temps t+1 est simplement d'étendre le chemin initial (définition récursive).

b) Calcul des probabilités d'avoir un symbole de sortie donné : forward algorithm

Nous commençons par une définition :

Equation A-13

$$\alpha_i(t) \overset{def}{=} P(w_{1,t-1}, s_t = s^i), t > 1$$

$\alpha_i(t)$ représente la probabilité de produire la séquence de symboles $w_{1,t-1}$ pour se terminer dans l'état s^i. Lorsque t=1, nous pouvons considérer $w_{1,0}$ comme étant la chaîne vide. De cette remarque, nous pouvons alors donner une définition de $\alpha_i(1)$:

Equation A-14

$$\alpha_i(1) \overset{def}{=} \begin{cases} 1.0 \text{ si } i = 1 \\ 0 \text{ sinon} \end{cases}$$

Dans ce cas on incorpore le fait que nous devons commencer dans l'état s^1 (état initial). La quantité $\alpha_i(t)$ est appelée la *probabilité en avant (forward probability)*. Ce nom vient du fait qu'on calcule la probabilité en se déplaçant vers l'avant dans la séquence.

On peut calculer $P(w_{1,n})$ en fonction de $\alpha_i(t)$:

Equation A-15

$$P(w_{1,n}) = \sum_{i=1}^{\sigma} P(w_{1,n}, s_{n+1} = s^i)$$
$$= \sum_{i=1}^{\sigma} \alpha_i(n+1)$$

Ensuite nous pouvons calculer $\alpha_j(t+1)$'s à partir des $\alpha_j(t)$'s :

Equation A-16

$$\begin{aligned}
&\alpha_j(t+1) = P(w_{1,t}, s_{t+1} = s^j) \\
&= \sum_{i=1}^{\sigma} P(w_{1,t}, s_t = s^i, s_{t+1} = s^j) \\
&= \sum_{i=1}^{\sigma} P(w_{1,t-1}, s_t = s^i) P(w_t, s_{t+1} = s^j \mid w_{1,t-1}, s_t = s^i) \\
&= \sum_{i=1}^{\sigma} P(w_{1,t-1}, s_t = s^i) P(w_t, s_{t+1} = s^j \mid s_t = s^i) \\
&= \sum_{i=1}^{\sigma} \alpha_i(t) P(s^i \xrightarrow{w_t} s^j)
\end{aligned}$$

Cet algorithme permet donc de calculer assez simplement la probabilité qu'une certaine séquence de symboles apparaisse. Comme on l'a déjà dit, selon l'utilisation qu'on veut faire du modèle, les symboles sont des lettres, des mots...

c) Calcul des probabilités d'avoir un symbole de sortie donné : backward algorithm

Cet algorithme permet aussi de calculer les probabilités de générer certains symboles de sortie. La différence avec le précédent réside dans le fait qu'on part de la fin de la séquence générée.

On a donc :

Equation A-17

$$\beta_i(t) \overset{def}{=} P(w_{t,n} \mid s_t = s^i)$$

On calcule les même quantités que pour la forward probability :

Equation A-18

$$\beta_1(1) = P(w_{1,n} \mid s_1 = s^1) = P(w_{1,n})$$

On note que la probabilité de produire le symbole vide , au temps n+1 (lorsque toute la chaîne d'entrée est lue) est 1. On a alors

Equation A-19

$$\beta_i(n+1) = P(\varepsilon \mid s_{n+1} = s^i) = 1$$

Voici enfin l'étape principale de notre algorithme (récursive) :

Equation A-20

$$
\begin{aligned}
&\beta_i(t-1) = P(w_{t-1,n} \mid s_{t-1} = s^i) \\
&= \sum_{j=1}^{\sigma} P(w_{t-1,n}, s_t = s^j \mid s_{t-1} = s^i) \\
&= \sum_{j=1}^{\sigma} P(w_{t-1}, s_t = s^j \mid s_{t-1} = s^i) P(w_{t,n} \mid w_{t-1}, s^t = s^j, s_{t-1} = s^i) \\
&= \sum_{j=1}^{\sigma} P(w_{t-1}, s_t = s^j \mid s_{t-1} = s^i) P(w_{t,n} \mid s_t = s^j) \\
&= \sum_{j=1}^{\sigma} P(s^i \xrightarrow{w_{t-1}} s^j) \beta_j(t)
\end{aligned}
$$

Maintenant que nous avons ces deux algorithmes *forward* and *backward*, nous pouvons essayer de résoudre un problème important : comment attribuer les paramètres aux arcs des chaînes de Markov d'une manière automatique? Il existe pour cela un algorithme, le *forward-backward algorithm*, décrit plus en détail dans **[Baum72]**, que nous introduisons dans la paragraphe suivant.

A.6. L'entraînement des chaînes de Markov

a) Introduction

Nous allons maintenant définir un algorithme qui étant donné une *séquence d'entraînement*, ajuste les probabilités des paramètres d'une chaîne de Markov cachée (HMM). Si cette séquence d'entraînement est représentative, elle améliore le comportement du modèle en affinant les paramètres. Comme nous ne connaissons pas réellement le comportement de la langue modélisée, nous allons utiliser un corpus, qui est un échantillon représentatif de la langue cible. Dans ce qui suit, nous commençons par expliquer la façon d'attribuer des paramètres à une chaîne de Markov. Ensuite, nous affinons les formules données pour pouvoir les utiliser sur des chaînes de Markov cachées (HMM).

b) Algorithme d'entraînement

Commençons par décrire un moyen d'attribuer les probabilités aux arcs d'une chaîne de Markov que voici :

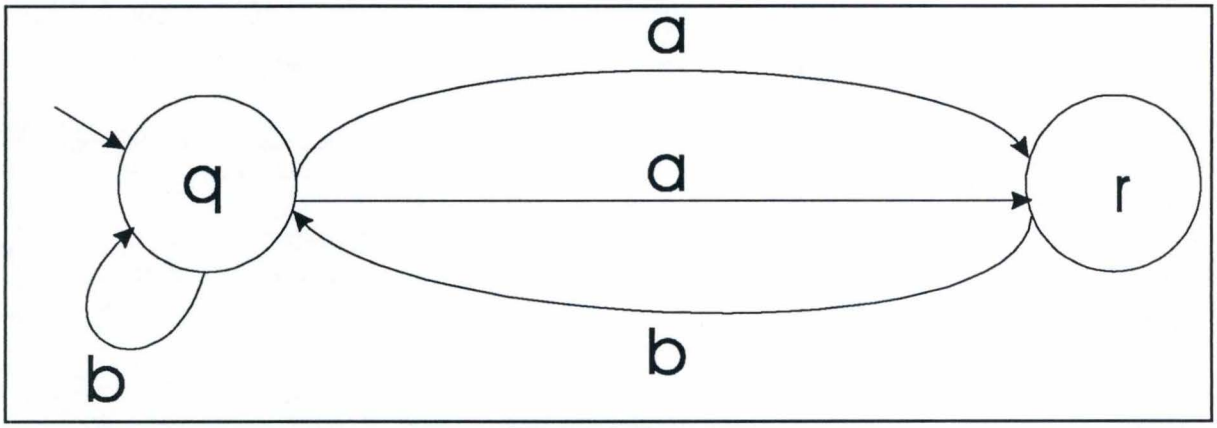


Figure A-5

Remarquons d'abord qu'il ne s'agit **pas** d'une chaîne de Markov cachée, ce qui simplifie le problème. Dans ce cas, il est assez simple de déterminer une façon d'attribuer des probabilités. Rappelons qu'avec une chaîne de Markov il est possible de déterminer la séquence d'états qui a permis de générer la séquence de sortie.

Le principe de l'algorithme est le suivant : on choisit une séquence de sortie qui soit représentative de la réalité (séquence d'entraînement). On essaye alors de déterminer la séquence d'états ayant permis de générer notre chaîne de sortie. On établit un comptage du nombre de transitions qui ont été utilisées, ce qui va permettre d'établir des probabilités d'une manière exacte.

Pour illustrer cet algorithme, voici un exemple concret, définis sur la chaîne de Markov du début de ce paragraphe (**figure A.5**). Supposons que notre séquence d'entraînement soit **abbaababbaaa**. Lorsqu'on suit cette séquence dans la chaîne de Markov, nous trouvons que chaque transition est générée un certain nombre de fois :

arc de	*arc vers*	*symbole généré*	*occurrences*
q	r	a	5
q	q	b	3
r	q	a	1
r	q	b	2

Les probabilités de chaque transition pourront alors être établies par une formule très simple :

Equation A-21

$$P\left(s^{i} \xrightarrow{w^{k}} s^{j}\right) = \frac{C\left(s^{i} \xrightarrow{w^{k}} s^{j}\right)}{\sum_{l=1,m=1}^{\sigma,\varpi} C\left(s^{i} \xrightarrow{w^{m}} s^{l}\right)}$$

Cette formule est à la base de l'algorithme d'entraînement décrit dans ce chapitre. Seule la façon de calculer les poids des transitions va subir quelques variations pour continuer d'utiliser cette formule lors de l'utilisation de chaînes de Markov cachées.

En appliquant cette formule, on aura par exemple $P_e(q \xrightarrow{a} r) = 5/8$, ou encore $P_e(q \xrightarrow{b} q) = 3/8$.

Les chaînes de Markov cachées (HMM) ont la particularité que plusieurs séquences d'états peuvent générer la même chaîne de sortie. Nous ne sommes donc pas capables d'établir avec certitude quelle transition a été utilisée à chaque étape. **[Charniak93]** insiste sur un principe à utiliser dans ce cas : si nous ne sommes pas certains de la transition utilisée lors d'une étape, nous supposons qu'elles sont toutes utilisées mais au prorata de la probabilité d'utilisation du chemin sur lequel elles se trouvent. Par exemple, si on a le choix entre deux transitions a et b respectivement sur des chemins ayant respectivement une probabilité de 1/3 et 2/3, alors on donne un poids de 1/3 à la transition a et 2/3 à la transition b. De plus, si une même transition apparaît plusieurs fois dans un chemin, on en tient compte en multipliant son poids par le nombre d'apparitions de cette transition. Le calcul du poids d'une transition est donc légèrement modifié.

Nous commençons par le décrire d'une manière informelle, pour faciliter sa compréhension. On calcule le nombre de fois où la transition est utilisée dans le chemin, on multiplie ce nombre par la probabilité d'emploi du chemin sur lequel la transition se trouve. On réalise ce calcul pour chaque chemin possible, et on additionne tous les résultats. Donnons maintenant cette définition d'une manière plus formelle :

Equation A-22

$$C(s^i \xrightarrow{w^k} s^j) = \sum_{S_{1,n+1}} P(S_{1,n+1} | W_{1,n}) \eta(s^i \xrightarrow{w^k} s^j, S_{1,n}, W_{1,n})$$

$$= \frac{1}{P(W_{1,n})} \sum_{S_{1,n+1}} P(S_{1,n+1}, W_{1,n}) \eta(s^i \xrightarrow{w^k} s^j, S_{1,n}, W_{1,n})$$

$\eta(s^i \xrightarrow{w^k} s^j, S_{1,n}, W_{1,n})$ compte le nombre de fois que $s^i \xrightarrow{w^k} s^j$ apparaît dans la séquence d'états $S_{1,n+1}$ quand la sortie est $W_{1,n}$.

Cet algorithme comporte un problème : le calcul de la probabilité de chaque arc nécessite de connaître la probabilité du chemin sur lequel chaque arc se trouve. Or connaître la nécessité du chemin nécessite la connaissance des probabilités de chaque arc. Pour résoudre ce problème, la méthode utilisée est de *deviner* les probabilités initiales des arcs, et on utilise ces probabilités dans l'**équation A.22**. Ces nombres seront alors utilisés pour calculer les nouvelles valeurs des probabilités par l'**équation A.21**. Nous utilisons ces nouvelles probabilités pour recalculer les paramètres (probabilités) à nouveau, et ce jusqu'à ce que les nombres se stabilisent. On peut maintenant donner un algorithme pour l'entraînement des chaînes de Markov :

Algorithme d'entraînement :

Deviner les paramètres de la HMM

Boucler jusqu'à ce que (anciens paramètres de la chaîne ≅ nouveaux paramètres de la chaîne)

anciens paramètres = nouveaux paramètres

nouveaux paramètres = calculer_paramètres()

Fin de boucle

Il n'est pas évident de se convaincre du fonctionnement de cet algorithme, même décrit informellement. Une description complète de celui-ci peut être trouvée dans **[Baum72] [Levinson et al. 83]**.

On constate que l'**équation A.22**, décrivant la manière dont on calcule le poids d'une transition, est très peu performante du fait qu'elle doit itérer sur l'ensemble des chemins possibles pour générer la séquence d'entraînement, qui généralement croît exponentiellement avec la longueur de cette dernière. Il est donc souhaitable d'améliorer les performances de cet algorithme.

La première étape est de modifier l'**équation A.22** en une forme qui n'itère pas sur tous les chemins possibles $S_{1,n+1}$.

Nous allons maintenant exposer une série d'équations permettant d'établir un algorithme plus performant :

Equation A-23

$$C(s^i \xrightarrow{w^k} s^j) = \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} \sum_{S_{1,n+1}} P(S_{1,n+1}, W_{1,n}) \delta_t(s^i \xrightarrow{w^k} s^j, S_{1,n+1}, W_{1,n})$$

$$= \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} \sum_{S_{1,n+1}} P(S_t = s^i, S_{t+1} = s^j, W_t = w^k, S_{1,n+1}, W_{1,n})$$

La fonction $\delta_t(s^i \xrightarrow{w^k} s^j, S_{1,n}, W_{1,n})$ renvoie la valeur 1 si la transition est utilisée au temps t et "0" sinon. Ainsi, plutôt que de calculer le nombre de fois que la transition est utilisée dans le chemin comme dans l'**équation A.22,** on vérifie si la transition est utilisée à chaque moment t.

La prochaine étape nécessite de transformer légèrement l'**équation A.8** :

Equation A-24

$$P(W_{1,n}, S_t = s^i, S_{t+1} = s^j, W_t = w^k) = \sum_{S_{1,n+1}} P(S_t = s^i, S_{t+1} = s^j, W_t = w^k, S_{1,n+1}, W_{1,n})$$

En substituant l'**équation A.24** dans l'**équation A.23** on obtient alors

Equation A-25

$$C(s^i \xrightarrow{w^k} s^j) = \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} P(S_t = s^i, S_{t+1} = s^j, W_t = w^k, W_{1,n})$$

Nous pouvons alors développer notre algorithme définitif de comptage des transitions (développé à partir de l'**équation A.25**):

Equation A-26

$$C(s^i \xrightarrow{w^k} s^j) = \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} P(S_t = s^i, S_{t+1} = s^j, W_t = w^k, W_{1,n})$$

$$= \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} P(W_{1,t-1}, S_t = s^i, W_t = w^k, S_{t+1} = s^j, W_{t+1,n})$$

$$= \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} P(W_{1,t-1}, S_t = s^i) P(W_t = w^k, S_{t+1} = s^j \mid W_{1,t-1}, S_t = s^i) P(W_{t+1,n} \mid W_{1,t}, S_t = s^i, S_{t+1} = s^j)$$

$$= \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} P(W_{1,t-1}, S_t = s^i) P(W_t = w^k, S_{t+1} = s^j \mid S_t = s^i) P(W_{t+1,n} \mid S_{t+1} = s^j)$$

$$= \frac{1}{P(W_{1,n})} \sum_{t=1}^{n} \alpha_i(t) P(s^i \xrightarrow{w^k} s^j) \beta_j(t+1)$$

Cette dernière équation nous est très utile puisque nous connaissons déjà les algorithmes pour calculer $\alpha_i(t-1)$ et $\beta_j(t+1)$. Aussi pour calculer les poids attribués à chaque transition, nous pouvons simplement utiliser l'**équation A.26**. Voici un exemple d'utilisation de cet algorithme appliqué à la chaîne de Markov de la **figure A.6**, et en utilisant la séquence d'entraînement « 01011 ». On effectue une première évaluation des paramètres (**fig. A.6 b**), et on va essayer d'approcher le plus possible les valeurs des paramètres réels (**fig. 6 a**). Voici ce modèle de Markov :

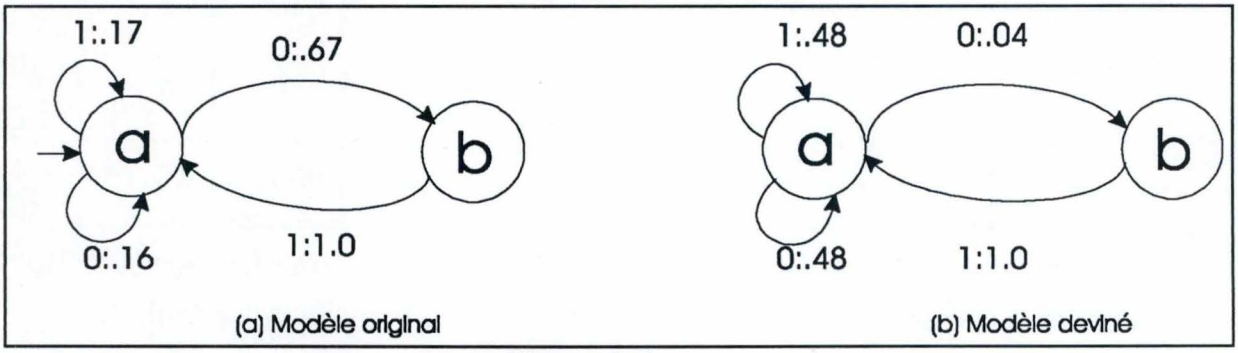


Figure A-6

Pour appliquer l'**équation A.26**, il est nécessaire de commencer par calculer les $\alpha_i(t)$ et les $\beta_j(t)$:

	1	2	3	4	5	6
	ϕ	0	1	"0"	1	1
$\alpha_a(t)$	1	.48	.27	.13	.072	.035
$\alpha_b(t)$	0	.04	0	.01	0	0

	1	2	3	4	5	6
	0	1	0	1	1	ϕ
$\beta_a(t)$	.035	.062	.13	.23	.48	1
$\beta_b(t)$	0	.13	0	.28	1	1

On utilise alors les résultats obtenus par l'**équation A.26** dans l'équation 21pour obtenir les probabilités d'utilisation d'une transition :

	0	1	0	1	1	Total	New P
$a \xrightarrow{0} b$	.0052	0.0	.0052	0."0"	0.0	.010	**.06**
$b \xrightarrow{1} a$	0.0	.0052	0.0	.0048	0.0	.010	**1.0**
$a \xrightarrow{0} a$	.030	0.0	.030	0."0"	0.0	.060	**.36**
$a \xrightarrow{1} a$	0.0	.030	0.0	.030	.035	.095	**.58**

On constate que les nouvelles probabilités déterminées après cette première itération de l'algorithme sont déjà plus proches des probabilités réelles.

Cet algorithme permet donc réellement d'apprendre le langage, et de l'affiner sans cesse, et ce d'une manière entièrement automatique, ce qui

n'est pas négligeable. L'algorithme n'est cependant pas parfait et comporte certains défauts **[Charniak93 b]**.

De nombreuses utilisations peuvent être faites d'un tel modèle. Nous nous concentrerons dans le chapitre V. sur des utilisations possibles de l'apprentissage statistique du langage dans des applications de correction orthographique et de génération de langage. D'autres possibilités existent encore. Par exemple, nous pourrions construire un modèle de Markov qui correspondrait à la grammaire d'un langage. Chaque état serait alors une catégorie syntaxique, et chaque transition comporterait un symbole de sortie étant un mot. Nous expliquons cet exemple ci-dessous dans le but de montrer un exemple plus concret de l'utilisation des chaînes de Markov cachées.

A.7. Un exemple concret

Ce paragraphe expose un exemple plus concret découlant de l'utilisation du modèle de Markov, qui consiste en l'attribution des catégories syntaxiques des mots d'un corpus par le modèle.

Pour cela, nous pouvons considérer les catégories syntaxiques comme étant les états de la chaîne. Les symboles de sorties sont les mots de la langue. Nous pouvons alors réécrire l'**équation A.8** en tenant compte de ces nouveaux paramètres :

Equation A-27

$$P(w_{1,n}) = \sum_{t_{1,n+1}} P(w_{1,n}, t_{1,n+1})$$

Dans ce cas, $t_{1,n+1}$ est une séquence de *n+1* catégories syntaxiques (*tags*). Nous interprétons cette séquence comme les catégories syntaxiques des n mots correspondants à $w_{1,n}$. La dernière catégorie syntaxique est considérée comme une prédiction du prochain *tag* .

Nous pouvons maintenant définir le problème de l'assignation des catégories syntaxiques comme étant :

Equation A-28

$$\arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \arg\max_{t_{1,n}} \frac{P(w_{1,n},t_{1,n})}{P(w_{1,n})}$$
$$= \arg\max_{t_{1,n}} P(w_{1,n},t_{1,n})$$

où $\arg\max_{x} f(x)$ est la valeur de x qui maximise f(x). Aussi nous désirons déterminer la valeur de $t_{1,n}$ qui maximise $P(t_{1,n},w_{1,n})$.

Les équations correspondant à notre modèle du langage peuvent être dérivées de l'**équation A.27** ainsi que de deux hypothèses :

1. La probabilité d'apparition d'un mot à une certaine position, étant donné sa catégorie syntaxique est indépendante de tout autre événement.
2. La probabilité qu'une certaine catégorie syntaxique apparaisse pour le mot suivant est dépendante seulement de la catégorie syntaxique du mot présent.

Ces équations sont alors :

Equation A-29

$$P(w_n|w_{1,n-1},t_{1,n}) = P(w_n|t_n)$$ et

Equation A-30

$$P(t_n|w_{1,n-1},t_{1,n-4}) = P(t_n|t_{n-1})$$

L'**équation A.27** devient alors :

Equation A-31

$$P(w_{1,n}) = \sum_{t_{1,n+1}} P(w_{1,n},t_{1,n+1})$$
$$= \sum_{t_{1,n+1}} \prod_{i=1}^{n} P(w_i|t_i)P(t_{i+1}|t_i)$$

En s'aidant de ce modèle, et des algorithmes décrits dans ce chapitre, nous pouvons obtenir la séquence des catégories syntaxiques (la plus probable) auxquelles chaque mot appartient.

D'autres modélisations sont bien sûr possibles. Ainsi, nous pouvons émettre l'hypothèse qu'un état représente deux catégories syntaxiques : celle du mot présent et celle du mot précédent. Il faut cependant être très prudent lors du choix des éléments conditionnels.

Nous n'avons pas encore abordé le problème de la structure des chaînes de Markov. Celle-ci peut bien sûr être construite manuellement, et affinée au fur et à mesure de l'utilisation au fil de l'expérience. Un problème avec cette solution est qu'à chaque modification de la structure de la chaîne, il faut recommencer tout l'apprentissage, ce qui peut demander un temps relativement long, suivant les méthodes utilisées.

Des méthodes automatiques peuvent prendre le relais. Par exemple, le processus de classification (*clustering*)[5] permet de diviser l'ensemble des mots en classes. Si le processus a été bien paramétré, les classes correspondront par exemple aux catégories syntaxiques principales. Chaque classe pourrait alors correspondre à un état de la chaîne de Markov.

A.8. L'évaluation d'un modèle

Il serait très intéressant de pouvoir comparer des modèles entre eux, et de pouvoir déterminer lequel est le plus proche de la réalité. Pour cela, **[Charniak93]** introduit les notions d'entropie et d'entropie croisée, que nous exposons brièvement dans ce chapitre.

a) Longueur moyenne d'un message

Supposons que nous ayons à envoyer un grand nombre de messages. Chacun d'entre eux est choisi au sein d'un ensemble fini avec une probabilité connue.

Imaginons encore qu'un message est envoyé toutes les minutes et indique si une maison contient 0,1 ou 2 occupants à ce moment :

---

[5] *Voir chapitre III.B*

Situation	*Probabilité*	*Code*
Aucun occupant	0.5	0
Premier occupant	0.125	110
Second occupant	0.125	111
Deux occupants	0.25	10

Tableau A-1

On constate que la longueur des messages est variable, et qu'elle est fonction de la probabilité d'apparition de chaque message. On peut alors très simplement calculer la longueur moyenne d'un message qui est

$$\frac{1}{2} * 1 \text{ bit} + \frac{1}{4} * 2 \text{ bits} + \frac{1}{8} * 3 \text{ bits} + \frac{1}{8} * 3 \text{ bits} = 1.75 \text{ bits}.$$

b) Notion d'entropie

On peut voir un message comme une variable aléatoire W qui peut prendre une des valeurs d'un ensemble $V(W)$ (quatre dans notre exemple) avec une probabilité de distribution P.

[Charniak93] définit l'entropie d'une variable aléatoire comme :

Equation A-32

$$H(w) \overset{def}{=} -\sum_{w \in V(W)} P(w) \log P(w)$$

Cette formule reflète en fait le nombre moyen de bits à envoyer par message. Il est aussi possible d'interpréter cette formule comme une mesure d'incertitude du message. Reprenons notre exemple du paragraphe *a)* pour éclaircir ce point. L'entropie du message W est $H(W) = -(\frac{1}{2}\log\frac{1}{2} + \frac{1}{4}\log\frac{1}{4} + \frac{1}{8}\log\frac{1}{8} + \frac{1}{8}\log\frac{1}{8}) = 1.75$. Supposons maintenant que la probabilité de chaque message possible soit plus élevée. Par exemple nous pouvons avoir :

Situation	*Probabilité*	*Code*
Aucun occupant	0.5	0
Occupants	0.5	1

Tableau A-2

Il est évident que dans ce cas l'entropie du message est moins élevée :

$-(\frac{1}{2}\log\frac{1}{2}+\frac{1}{2}\log\frac{1}{2})=1$. Nous pouvons donc interpréter cela comme le fait que notre message peut être compris avec plus de certitude que le précédent.

Il est également possible d'adapter l'**équation A.32** au calcul de l'entropie d'une séquence de mots. On a alors

Equation A-33

$$H(w_{1,n}) = -\sum_{w_{1,n}\in V(w_{1,n})} P(w_{1,n})\log P(w_{1,n})$$

On peut écrire cette dernière équation d'une manière plus concise en remplaçant $\sum_{w_{1,n}\in V(w_{1,n})}$ par $\sum_{w_{1,n}}$, ce qui donne :

Equation A-34

$$H(w_{1,n}) = -\sum_{w_{1,n}} P(w_{1,n})\log P(w_{1,n})$$

Comme il peut ne pas être évident de comparer l'entropie de messages de longueurs différentes (car l'entropie augmente avec la longueur du message), on utilise parfois l'entropie par mot :

Equation A-35

$$\frac{1}{n}H(w_{1,n}) = -\frac{1}{n}\sum_{w\in V(W)} P(w_{1,n})\log P(w_{1,n})$$

où n représente la longueur du message.

Nous pouvons donc déterminer si un message est plus ou moins incertain par rapport à d'autres messages. Dans la suite de ce chapitre, nous expliquons comment généraliser cette définition d'incertitude d'un message

à celle d'incertitude d'un modèle du langage. Nous voyons alors comment déterminer la supériorité d'un modèle de Markov par rapport à un autre.

c) Notion d'entropie croisée

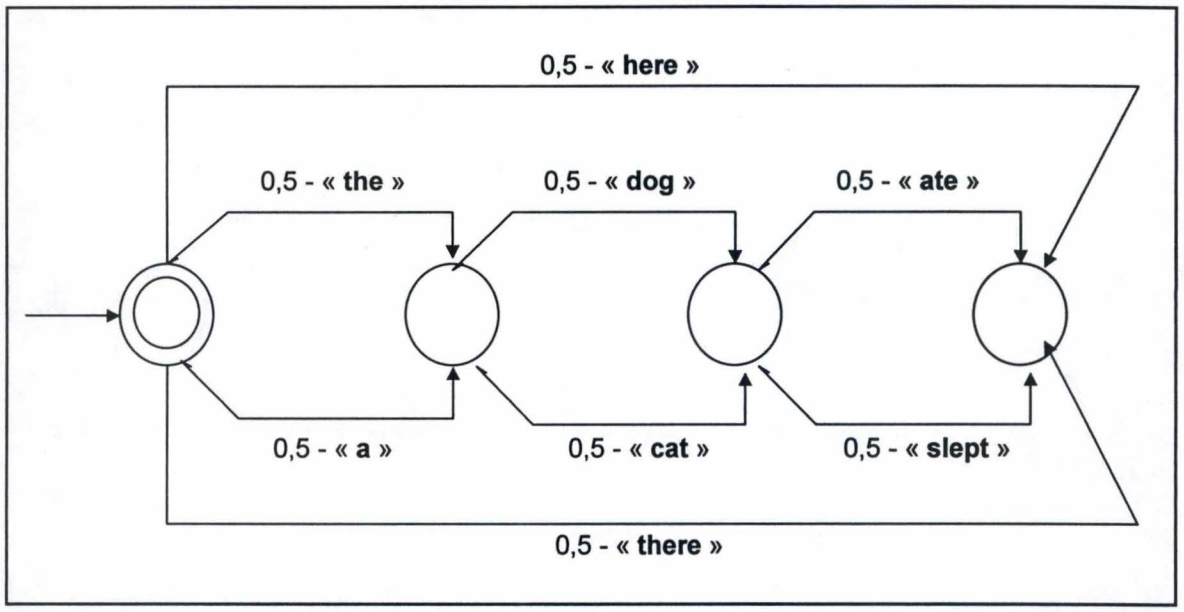


Figure A-7

Supposons que ce modèle corresponde au langage que nous voulons modéliser. Nous ne le connaissons pas et travaillons avec un modèle beaucoup plus simple, qui consiste simplement à choisir des mots au hasard parmi un ensemble de huit mots. Nous pouvons alors nous demander « *Quelle est la qualité de notre modèle ?* » (en supposant qu'il existe un seul modèle correct). Une façon de répondre à cette question est de calculer l'entropie croisée définie comme suit :

Equation A-36

$$H(W_{1,n}, P_M) \overset{def}{=} -\sum_{W_{1,n}} P(W_{1,n}) \log P_M(W_{1,n})$$

L'entropie croisée par mot est donnée par

Equation A-37

$$\frac{1}{n} H(W_{1,n}, P_M) \overset{def}{=} -\frac{1}{n} \sum_{W_{1,n}} P(W_{1,n}) \log P_M(W_{1,n})$$

Ainsi l'entropie croisée du langage défini par la **Figure A.7** par rapport au choix aléatoire d'un mot est 3. Supposons maintenant que nous utilisons le même modèle de Markov que dans la **Figure A.7**, mais avec des

probabilités inexactes, à savoir que les probabilités arrivant et partant de chaque état ne sont plus 0.5 mais 0.75 et 0.25. L'entropie croisée entre les deux modèles est alors de 1.2 bits.

Comme nous pouvons le constater, l'entropie croisée d'un modèle incorrect est toujours plus élevée que celle du modèle correct.

Formellement, nous avons donc :

Equation A-38

$$\boxed{H(W_{1,n}) \leq H(W_{1,n}, P_M)}$$

L'égalité $H(W_{1,n}) = H(W_{1,n}, P_M)$ survient uniquement si $P = P_M$. Cela suggère que l'on peut donc mesurer l'entropie croisée de plusieurs modèles concurrents pour déterminer celui qui est le plus proche de la réalité.

Il subsiste cependant un problème : comment calculer l'entropie croisée puisqu'on ne connaît pas $W_{1,n}$ (puisqu'il s'agit de la probabilité d'apparition d'une séquence de mots dans le modèle réel, que nous ne connaissons pas). Ce problème est résolu en considérant qu'un corpus est un échantillon représentatif de la langue. Les probabilités d'apparition des séquences de mots dans le langage « réel » sont celles du corpus utilisé.

B. Classification

B.1. Introduction

La classification (ou clustering) étant un des principaux sujets de recherche menés par David M. W. Powers, celui-ci a tenu à ce que cette méthode soit étudiée d'une manière générale en vue d'une possibilité d'utilisation de celle-ci dans le cadre du développement de notre application.

Le classification est un processus de partitionnement d'un ensemble d'objets en plusieurs groupes. Dans ce chapitre, nous discutons d'apprentissage non supervisé, c'est à dire que le système n'exige pas de « professeur » pour pré-classifier les objets, mais utilise une fonction d'évaluation pour découvrir les classes. Le but est donc de grouper ces objets en classes (ou *clusters*) de telle manière que les objets repris dans une même classe aient un grand degré d'***association naturelle*** tandis que les classes restent ***relativement distinctes*** entre elles (**[Bow92]**).

Le processus de classification se présente comme suit : nous avons un ensemble d'objets dénotés $x_1, x_2, ..., x_m$ repris dans un espace S. Il nous faut partitionner cet espace S en $s_1, s_2, ..., s_k$ de telle manière que :

- $s_1 \cup s_2 \cup s_3 .... \cup s_k = S$
- $s_i \cap s_j = \varnothing \quad \forall i \neq j$

La solution à ce problème peut ne pas être unique, puisque plus d'un partitionnement pourrait être généré. Donc, le critère pour identifier les clusters dépend du choix que l'on donnera à la signification des termes « association naturelle » et « relativement distinct ».

La procédure générale à suivre pour modéliser cette classification est la suivante (**[Charniak93d]**) :

1. Définir les propriétés des objets à grouper et être capable de donner des valeurs numériques à ces propriétés.
2. Créer les vecteurs de longueur N comprenant comme éléments les N valeurs numériques.
3. Visualiser les vecteurs à N dimensions comme des points dans un espace et grouper les points qui sont proches (mesure de distance).

Trois variations sont possibles lors de l'application de cette procédure :

- Les propriétés utilisées pour définir le vecteur.
- La mesure de distance (ou mesure de similarité) utilisée pour décider si deux points sont proches.
- L'algorithme utilisé pour réaliser la classification.

B.2. Objets

Pour calculer la similarité entre les objets, il est donc intéressant de connaître la nature exacte de ceux-ci qui peuvent être des phonèmes, des lettres, des mots, ... Remarquons que dans notre cas, nous voulons décrire une classification de mots. C'est la raison pour laquelle nous nous servons du contexte comme propriété. Le contexte d'un mot est défini comme étant un mot ou une séquences de mots qui le suivent ou le précèdent directement. Nous modélisons la propriété d'un mot comme un vecteur. Chaque élément de ce vecteur représente une relation entre le mot étudié (*mot de concept*) et un autre mot (*mot de contexte*) du corpus. La valeur de cet élément nous donne le nombre de fois que le *mot de contexte* se trouve directement placé à côté du *mot de concept*.

De manière plus pratique, nous avons un corpus qui représente un ensemble de mots. Différents mots et combinaisons de mots ont lieu selon différentes fréquences ; fréquences qui sont caractéristiques du langage. Cela signifie que nous pouvons produire des statistiques à partir des séquences de mots qui se présentent le plus souvent. Les statistiques de fréquences pour une séquence de N mots sont appelées n-*grammes*, et les *bigrammes* et *trigrammes* correspondent respectivement aux paires et triplets de mots. Nous travaillons avec des *bigrammes* (un mot représentant le concept et un mot représentant le contexte droit). Donc, si nous prenons la phrase : « Alice in Wonderland. », nous pouvons dégager les deux *bigrammes* suivants : (Alice-in), (in-Wonderland). Le *bigramme* (Alice-in) est composé d'un concept (mot « Alice ») et son contexte droit (mot « in »). Remarquons que dans notre cas, nous avons choisi de ne pas tenir compte des caractères de ponctuation et des blancs.

Il est possible de visualiser la classification de mots comme un problème de classification de points dans un espace à n-dimensions. Les points sont positionnés dans cet espace à n-dimensions. Un point représente la propriété d'un mot du corpus, définie par les fréquences de celui-ci avec chaque autre mot du corpus se plaçant directement à sa suite (contexte droit). Un exemple décrivant cette technique se trouve ci-dessous.

Pour modéliser la propriété utilisée, nous utilisons une matrice que nous appelons *matrice conceptuelle* (**[Powers97]**). Pour un *bigramme*, nous plaçons le concept en ordonnée, le contexte en abscisse et la fréquence de ce *bigramme* comme valeur (voir tableau ci-dessous pour le texte : « *Alice in Wonderland. Alice has a rabbit. A rabbit is an animal.* »).

Concept/Contexte	*Alice*	*in*	*Wonderland*	*has*	*is*	*an*	*a*	*animal*	*rabbit*
Alice	0	1	0	1	0	0	0	0	0
in	0	0	1	0	0	0	0	0	0
Wonderland	0	0	0	0	0	0	0	0	0
has	0	0	0	0	0	0	1	0	0
is	0	0	0	0	0	1	0	0	0
a	0	0	0	0	0	0	0	0	2
an	0	0	0	0	0	0	0	1	0
animal	0	0	0	0	0	0	0	0	0
rabbit	0	0	0	0	1	0	0	0	0

Tableau B-1 Un exemple de matrice conceptuelle

Nous avons autant de vecteurs contextuels[6] que de lignes dans cette matrice. Dès lors, le mot (concept) sera représenté dans un espace à n-dimensions par son vecteur contextuel. Chaque vecteur contextuel représente l'objet qui sera utilisé dans les différentes méthodes de classification que nous verrons par la suite.

B.3. Mesures de similarité ou de distance

Une mesure de similarité est habituellement donnée sous forme numérique pour indiquer le degré d'association naturelle ou le degré de ressemblance entre les objets d'un groupe, entre un objet et un groupe d'objets ou encore entre différents groupes d'objets. Différentes mesures de distance ont été proposées par les mathématiciens. Celles-ci peuvent être utilisées pour identifier une certaine similarité entre vecteurs. Voici les mesures de distance les plus courantes, reprises par **[Bow92]**, **[Finch93]**, **[Korkmaz97]** et **[Schifferdecker94]** :

[6] *Le vecteur contextuel pour le mot « Alice » sera (0,1,0,1,0,0,0,0,0).*

- ***Manhattan*** qui calcule la différence absolue entre les éléments des deux vecteurs. Cette mesure est définie comme suit :

Equation B-1

$$D(x,y) = \left(\sum_i |x_i - y_i| \right) \quad \text{tel que } (1 <= i <= n)$$

- ***Eucledian*** qui est définie comme suit :

Equation B-2

$$D(x,y) = \sqrt{\sum_i (x_i - y_i)^2} \quad \text{tel que } (1 <= i <= n)$$

- ***Spearman Rank Correlation Coefficient*** : qui est basée sur la différence entre les « rangs » de deux vecteurs plutôt que sur la différence entre les éléments. Cette mesure est définie comme suit (**[Korkmaz97]**) :

Equation B-3

$$D(x,y) = \sum_i \left(R(i,x) - R(i,y) \right)^2 \quad \text{tel que } (1 <= i <= n)$$

x et y sont les vecteurs et R(i,x) et R(i,y) sont les « rangs » des vecteurs correspondants. Dans notre cas, le « rang » est calculé en disposant les vecteurs dans l'intervalle [0,1]. Le composant avec la plus grande valeur parmi les composants du vecteur prend la valeur **1**. S'il y a *n* composants dans le vecteur, celui avec la seconde plus grande valeur correspond à **1-** *(1/n)* et ainsi de suite. La plus petite valeur correspond à zéro.

Les deux premières mesures présentent un inconvénient majeur car elles dépendent de la différence absolue entre les valeurs des composants du vecteur. En effet, bien que des mots appartiennent à la même catégorie syntaxique, les fréquences peuvent être totalement différentes. Par exemple, le mot « *go* » et le mot « *appreciate* » font partie de la même catégorie syntaxique et doivent être classifiés ensemble. Or, la fréquence de « *go* » est très élevée par rapport à celle de « *appreciate* ». Donc, si nous utilisons une mesure de distance basée seulement sur des différences absolues de vecteurs (Manhattan ou Eucledian), la distance calculée entre les fréquences élevées et les fréquences faibles sera haute, ce qui n'est pas désiré.

Il est clair que le degré de similarité sera haut parmi les objets appartenant à la même classe et bas pour les objets de classes différentes.

B.4. Méthodes de classification

Les méthodes de classification peuvent être vues comme une partition d'un ensemble d'éléments en un nombre de clusters. On peut dire que la plupart des méthodes de classification existantes se divisent en deux catégories :

- Méthodes hiérarchiques
- Méthodes de partitionnement

a) Méthodes hiérarchiques

Ces méthodes consistent à construire des clusters graduellement en mettant les objets les plus similaires ensemble et en produisant une hiérarchie de partitionnements qui peut être représentée par des arbres ou des dendrogrammes (voir **Figure B.1**). Le nombre de clusters à l'arrivée n'est pas spécifié lors du début de l'exécution de la méthode. Ces diagrammes nous permettent d'examiner la structure interne des clusters et la manière dont les clusters sont dérivés lors de l'exécution de l'algorithme.

Voyons l'algorithme réalisant une classification hiérarchique :

HIERARCHIQUE(O_1,...,O_n)
Données : ensemble de N objets
Résultats : hiérarchisation des N objets

$C = \{c(O_1),...,c(O_n)\}$
WHILE $|C| > 1$ DO
    Calculer (i,j) tel que $D(C_i,C_j)$ soit minimale
    $C = (C \setminus \{c(C_i),c(C_j)\}) \cup \{c(C_i,C_j)\}$
END

Le principe général de l'algorithme est donc le suivant : tant qu'il reste plus d'un cluster, il faut fusionner les deux clusters les plus proches selon la mesure de similarité choisie. Il y a au départ autant de clusters que d'objets ($C = \{c(O_1),...,c(O_n)\}$). Lors de chaque passage dans la boucle, une même série d'opérations va être exécutée, c'est-à-dire : calcul des distances entre les différents clusters, choix des clusters les plus similaires, fusion des ces deux clusters pour former un nouveau cluster et suppression des clusters fusionnés.

A partir du deuxième passage, il faut recalculer les distances par rapport au nouveau cluster créé. Les variantes possibles de cet algorithme ont lieu dans

cette étape. On remarque deux groupes de méthodes possibles **([Schifferdecker94])** :

- ***Centroïd***[7] : on prend les éléments dans les clusters à fusionner et on calcule un cluster de centre à partir duquel les nouvelles distances seront calculées. Nous avons deux clusters de centre x et y qui vont être fusionnés pour former un nouveau cluster de centre. Nous avons également P(x) et P(y), les probabilités[8] d'apparition pour les clusters x et y; et ω la fonction de normalisation définie comme suit :

Equation B-4

$$\omega(x) = \sum_{i=1}^{n} |x_i|$$

Il existe deux variantes pour calculer ce nouveau cluster de centre :

◊ avec poids :

Equation B-5

$$xy = \left(\frac{(P(x) * x) + (P(y) * y)}{\omega(P(x) * x) + (P(y) * y)} \right)$$

◊ sans poids :

Equation B-6

$$xy = \frac{x + y}{\omega(x + y)}$$

Lorsque le nouveau cluster de centre est déterminé, on calcule les distances avec tous les autres clusters de centre en se servant de la formule de l'étape 2, c'est à dire des mesures de distance.

- ***Récursive*** : on calcule les nouvelles distances récursivement à partir du nouveau cluster formé et avec l'aide des distances des deux clusters qui ont été fusionnés. Cela veut dire que la mesure de distance entre les vecteurs est seulement utilisée dans l'étape 2. Nous avons un cluster aléatoire z et xy, le nouveau cluster formé. Nous avons défini 4 méthodes pour déterminer la distance entre le nouveau cluster xy et le cluster z :

---

[7] *Terme anglais*

[8] *Nombre d'éléments du cluster divisé par le nombre total des éléments de l'ensemble des clusters.*

◊ le plus proche voisin :

Equation B-7

$$D(xz,y) = \min\left(D(x,z), D(y,z)\right)$$

◊ le voisin le plus éloigné :

Equation B-8

$$D(xy,z) = \max\left(D(x,z), D(y,z)\right)$$

◊ la distance moyenne :

Equation B-9

$$D(xy,z) = \frac{D(x,z) + D(y,z)}{2}$$

◊ la distance moyenne pondérée :

Equation B-10

$$D(xy,z) = \left(P(x) * D(x,z)\right) + \frac{P(y) * D(y,z)}{P(x) + P(y)}$$

b) Méthodes de partitionnement

Les méthodes de partitionnement offrent une autre technique pour générer des clusters. On définit préalablement le nombre K de clusters désirés, mais évidemment pas leur distribution des objets en clusters. C'est un problème d'optimisation que de chercher la partition de N objets en K clusters. Les méthodes de partitionnement consistent à diviser l'ensemble initial d'objets en K clusters (K étant donné). L'ensemble d'objets et l'ensemble de clusters vides sont les paramètres de l'algorithme présenté ci-dessous.

PARTITIONNEMENT(O_1,...,O_n)
Données : ensemble de N objets
Résultats : ensemble de K clusters

t = 0
$\{C_{1,t},...,C_{k,t}\}$ = INIT($\{O_1,...,O_n\}$)
DO
    Incrémenter t
    FOR EACH cluster C_p ($p = 1...K$) DO
        $S_p = \{O_n \mid D(O_n,C_{p,t-1}) < D(O_n,C_{j,t-1})\}$ $\forall j \neq p$
        $C_{p,t}$ = SELECTIONNER(S_p)
    END
WHILE $C_{k,t} \neq C_{k,t-1}$ $\forall k$
RETURN $\{C_{1,t},...,C_{k,t}\}$

Algorithme B-1 Partitionnement

Lors de la première itération (INIT), les objets sont répartis aléatoirement dans les clusters et un cluster de centre (voir paragraphe B.4.a.) est calculé pour chacun de ceux-ci. Ensuite, pour chaque cluster, nous calculons les distances de « similarité » entre celui-ci et les objets. La fonction notée $D(O_n,C_{p,t-1})$ désigne cette « similarité » qui se calcule à partir d'un vecteur (désignant l'objet) et d'un cluster de centre. Lors des itérations suivantes (SELECTIONNER), le cluster ($C_{p,t}$) est modifié sur base des objets qui lui ont été affectés (S_p) et un nouveau cluster de centre est calculé. La condition d'arrêt, quant à elle, dépend de la stabilité de la partition produite.

B.5. Visualisation

L'approche standard pour visualiser les classes linguistiques est le *dendrogramme*, l'arbre (**[Finch93]** et **[Schifferdecker94]**). Un exemple figure ci-dessous :

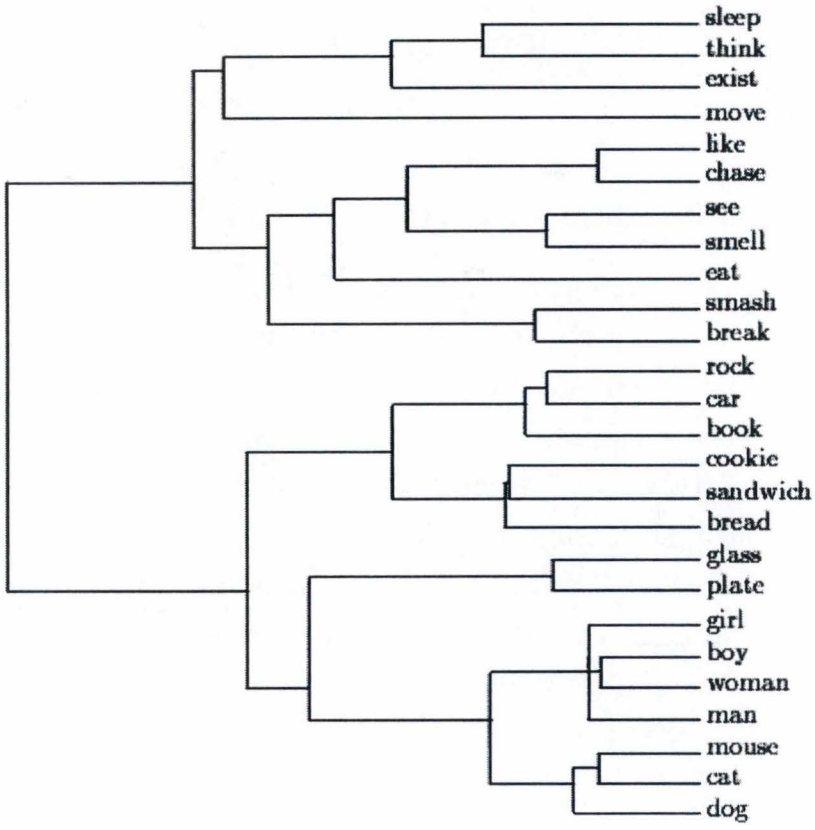


Figure B-1 Dendrogramme

Les *dendrogrammes* sont des arbres binaires formés en fusionnant[9] les plus proches voisins. La distance (voir paragraphe B.3.) entre les objets fusionnés est représentée par la longueur des branches du sous-arbre créé. Nous pouvons constater sur la figure ci-dessus que les branches les plus longues permettent de mettre en évidence deux classes : les verbes et les substantifs. Cette approche nous permet donc de visualiser les classes qui ont été générées lors de l'application de notre méthode de classification. Une autre technique pour valider le découpage de nos mots en classes syntaxiques est examinée dans le paragraphe suivant.

[9] *Résultat de la méthode de classification employée.*

B.6. Classification de mots

Comme nous l'avons vu précédemment, on peut déterminer préalablement le nombre de classes mais nous voulons trouver un groupement « naturel » des objets. C'est la raison pour laquelle, nous utilisons la méthode hiérarchique pour former des classes de mots selon leurs catégories syntaxiques. Cependant, cet algorithme s'arrête lorsqu'il reste une seule classe. Il faut donc trouver un moyen de mettre en évidence une classification valide avant la fin de l'algorithme. C'est pourquoi, **[Powers97]** a introduit une mesure supplémentaire (***goodness measure***) que nous appelons ***mesure de validité***. L'approche utilisée pour définir cette *mesure de validité* correspond à l'interprétation visuelle que nous adoptons pour inspecter les *dendrogrammes*. Sur base des expériences menées par David M. W. Powers (**[Powers97]**), nous définissons la *mesure de validité* comme le quotient entre le « fossé » autour de la classe et le diamètre de la classe. Le numérateur est défini comme la distance minimale entre un objet membre et un objet non membre d'une classe. Le dénominateur est défini comme la distance maximale entre deux objets membres d'une classe. Cette mesure est calculée à chaque itération de la méthode de classification. Voyons comment cela fonctionne en pratique. A une certaine itération, il est possible de fusionner la classe *A* et la classe *B* qui sont les plus « similaires ». La *mesure de validité* pour la classe *A* a été calculée et notée *mv(A)*. Cependant, *A* était composée de deux classes *A1* et *A2* (précédemment fusionnées) pour lesquelles nous avions calculé leurs *mesure de validité* : *mv(A1)* et *mv(A2)*. Si *mv(A)* est supérieure à *mv(A1)* ou *mv(A2)*, on peut dire que la classe *A* semble valide. Ensuite, nous calculons la mesure de similarité pour la classe *AB* (*mv(AB)*) résultant de la fusion des classes *A* et *B*. Si *mv(A)* est supérieure à *mv(AB)*, nous pouvons dire que la classe *A* est valide.

Les expériences menées par David M. W. Powers relatives à la définition d'une *mesure de validité* ainsi que les comparaisons effectuées entre les différentes mesures de distance sont disponibles dans **[Powers97]**.

B.7. Classification et langage naturel

La classification est fondamentale pour l'intelligence artificielle. En effet, un système qui ne serait capable d'induire un certain nombre de généralisations pertinentes sur la base des faits reçus serait rapidement noyé par une explosion de la quantité d'informations à prendre en compte. Les chercheurs rangent ce type de capacité dans la catégorie *apprentissage*, c'est-à-dire la capacité à améliorer son comportement au fil de l'expérience.

Dans notre cas, le phénomène d'apprentissage est présent. En effet, notre méthode de classification sans cesse appliquée à des corpora va nous permettre d'amplifier et d'affiner nos classes de mots. De plus, nous allons décrire deux façons de générer une grammaire (voir chapitre I.D.) sur base des classes de mots découvertes (**[Powers97]** et **[Finch93]**). Une première technique est l'***utilisation d'un lexique*** (par exemple *WordNet*[10]). Grâce à cela, les catégories syntaxiques de plusieurs mots sont à notre disposition. Il est donc possible de donner une catégorie syntaxique à chaque classe de mots. Ceci est basé sur l'hypothèse qu'au moins un mot de chaque classe est présent dans notre lexique. Ensuite, il nous reste à extraire chaque phrase de notre corpus et à dégager les règles de notre grammaire. Par exemple, si nous avons les classes de déterminants, d'adjectifs, de substantifs et de verbes, nous extrayons deux phrases : « *The black dog eats.* » et « *My son eats an apple.* ». Il est donc possible de déduire les règles suivantes :

R1 → Det Adj Nom Verbe
R2 → Det Nom Verbe Det Nom

Ensuite, nous supervisons ces règles et dégageons ***manuellement*** les syntagmes nominaux, les syntagmes verbaux, ... et nous avons une grammaire :

S → Np Vp
Np → Det Adj Nom
Np → Det Nom
Vp → Verbe
Vp → Verbe Det Nom

Muni de cette grammaire, nous pouvons construire un arbre de dérivation pour chaque phrase. Dans sa thèse, le Pr. Deville a créé des règles de dépendance qui permettent de retrouver les éléments grammaticaux d'une

[10] *Voir III.B.*

phrase sur base de l'arbre de dérivation découlant de celle-ci (**[Deville89]**). Ce principe nous facilite donc la recherche du sujet d'une phrase.

On en conclut qu'a partir d'un simple corpus, il nous est possible de créer des classes syntaxiques. Cependant, un inconvénient majeur se présente. Nous sommes incapables de lever l'ambiguïté syntaxique. En effet, l'analyse d'un corpus de langage naturel n'assigne pas toujours un mot à la même catégorie pour toutes ses occurrences ; c'est à dire que des mots peuvent remplir plusieurs rôles linguistiques. Par exemple, le mot « cut » peut être un nom (« *a nasty cut* »), un adjectif (« *loaves of cut and uncut bread* »), un infinitif (« *Mary told John to cut ant eat the cake* »), un verbe conjugué au passé (« *John cut and ate the cake* ») ou un participe passé (« *After John had cut and eaten the cake, Mary thanked him.* ») (**[Finch93]**).

Enfin, grâce à la classification de mots que nous avons décrite, nous sommes capables de générer une grammaire. Celle-ci deviendra de plus en plus précise au fur et à mesure que nous « entraînerons » notre classification sur d'autres corpus. Les classes syntaxiques trouvées ainsi que les règles de dépendance nous permettent d'identifier le sujet sont utiles pour l'implémentation de notre application. Cette question est abordée dans le chapitre consacré aux *Perspectives*.

C. Réseaux de neurones

C.1. Introduction

Dans ce chapitre, nous présentons un dispositif d'apprentissage calqué sur le fonctionnement réel du cerveau que nous avons brièvement abordé dans le chapitre II.E. Comme le cerveau, un réseau connexioniste (réseau neuronal) est constitué d'un grand nombre de cellules (neurones artificiels) reliées entre elles par des liens. Chaque neurone artificiel est capable d'influencer le comportement des neurones auxquels il est relié.

Voici une modélisation d'un neurone artificiel :

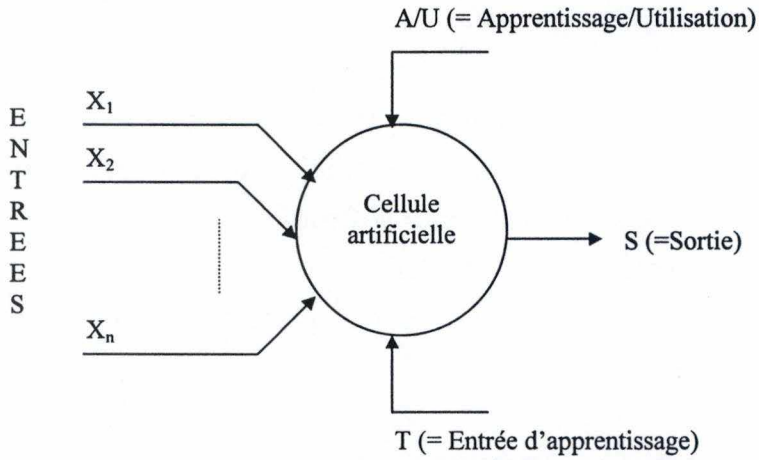


Figure C-1- Modélisation d'une cellule artificielle

Chaque connexion peut être dans l'état 0 ou 1. Les *n* entrées de la cellule représentent l'arrivée des synapses, et la sortie représente l'axone. Les entrées spéciales *Apprentissage/Utilisation* et *Entrée d'apprentissage* permettent d'entraîner le neurone. En effet, si l'entrée *A/U* est en mode *Apprentissage*, alors à chaque nouvelle présentation des entrées actuelles (masque d'entrée) en mode *Utilisation*, la sortie du dispositif émettra le signal présenté à l'entraînement (*entrée d'apprentissage noté T*).

Comme pour l'apprentissage statistique du langage, il est nécessaire d'entraîner le réseau à l'aide d'*exemples* (masques d'entrée associés à des sorties). Ceux-ci permettent d'associer des séquences particulières aux différentes entrées du réseau à des séquences particulières obtenues aux noeuds de sortie du réseau.

Une des caractéristiques principales des réseaux de neurones est leur capacité d'associer une séquence d'entrée qui n'a pas été présentée lors de l'entraînement, et de déterminer d'une manière automatique de quel exemple cette séquence se rapproche le plus.

De plus, un des avantages de ce dispositif est qu'il est très facile de le réaliser sous forme de composant électronique. Ainsi une puce électronique peut contenir un très grand nombre de cellules de base. Il est possible de réaliser des processeurs modélisant des réseaux de neurones, ce qui permet d'assez bonnes performances.

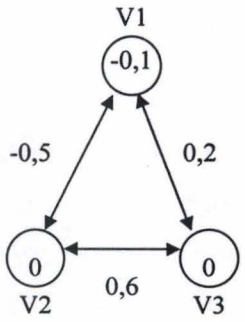
Nous présentons alors deux architectures de réseau neuronal (le modèle de Hopfield et le modèle multi-couches), et des algorithmes permettant d'utiliser ces deux modèles. Pour terminer ce chapitre, nous évoquons quelques domaines du traitement du langage naturel dans lesquels les réseaux de neurones pourraient être utilisés.

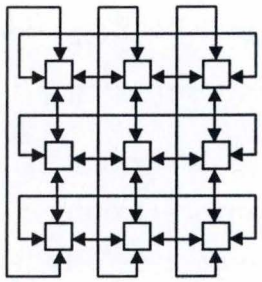
Notons que ce chapitre a pour but de décrire le fonctionnement général d'un réseau neuronal, et ne prétend pas solutionner l'entièreté des problèmes qui se présentent.

C.2. Modèle de Hopfield[11]

Le modèle de Hopfield décrit un réseau neuronal à une seule couche. L'unité de base d'un réseau de Hopfield est toujours le neurone artificiel. Chaque neurone V_i reçoit une entrée d'un neurone V_j avec une certaine force T_{ij} . S_i T_{ij} = 0, cela signifie que les neurones V_i et V_j sont déconnectés. On remarque que dans ce modèle $T_{ij}=T_{ji}$. Chaque neurone V_i peut générer une valeur 1 (il tire) ou 0 (il ne tire pas). Un neurone V_i est également défini par un seuil U_i. Pour qu'un neurone V_i tire, il faut que $\sum_{i \neq j} T_{ij} V_j > U_i$. Un réseau à une couche peut se présenter de différentes manières :

[11] *Source :* **[Aleksander et al. 90]**





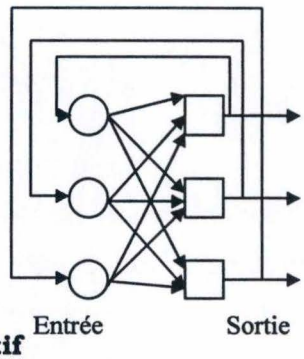


Figure C-2 - Réseau autoassociatif

Figure C-3 - Réseau simple

Figure C-4 - Réseau récurent

Un restriction dans le réseau de Hopfield est que chaque neurone tente de tirer avec une probabilité égale. Ainsi, on ne sait pas dire quel neurone va tirer le premier. Prenons le réseau simple et supposons qu'on soit dans l'état 000, c'est à dire qu'aucun neurone ne tire à cet instant. Si le neurone V_1 essaye de tirer, la somme de ses entrées est 0*(-0.5) + 0*(0.2) = 0 (>-0.1). Par contre, si V_2 essaye de tirer, la somme de ses entrées est seulement de 0*(-0.5)+0*(0.6) = 0 , ce qui n'est pas suffisant pour que le neurone tire. Le dernier neurone ne possède pas non plus une force suffisante pour tirer. On peut donc dire qu'il y a une probabilité de 1/3 pour que le système change d'état et de 2/3 pour qu'il reste dans le même état. Ci-dessous se trouvent le diagramme des états possibles du système et leurs probabilités d'apparition :

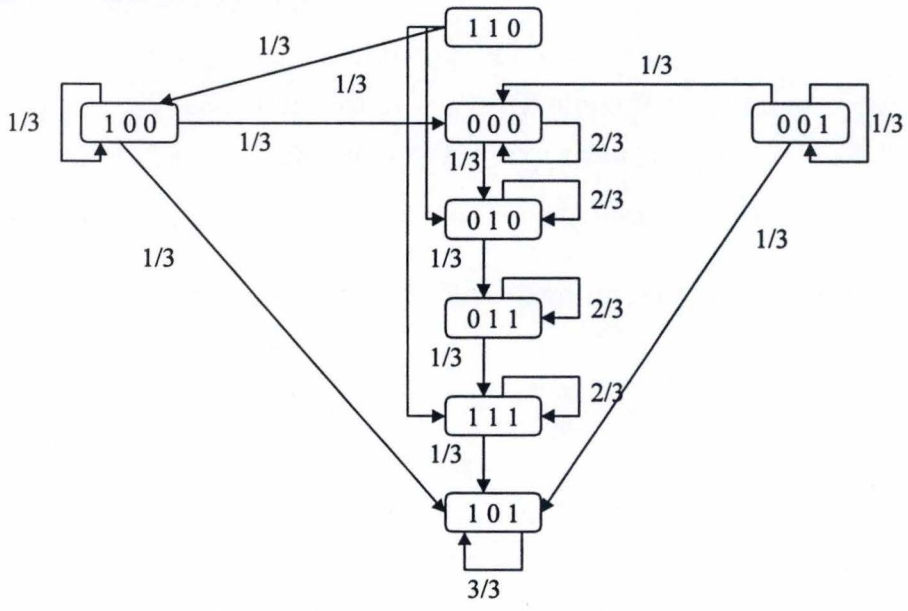


Figure C-5 Etats possibles et leurs transitions

On remarque que lorsque le système est dans l'état 101, il ne peut plus changer d'état (état **stable**). Cette constatation est la base de l'analyse de Hopfield.

Pour entraîner notre modèle, nous avons besoin de définir les états **stables**. Pour garantir ces états stables, il faut définir d'une certaine manière les seuils d'activation et la force avec laquelle les neurones interagissent. Basons-nous sur un exemple pour expliquer cette méthode.

- Supposons que nous désirons $V_1\ V_2\ V_3 = 010$ comme l'état stable A et $V_1\ V_2\ V_3 = 111$ comme l'état stable B.
- Pour l'état stable A, $V_1 = 0$ alors l'activation $T_{12}V_2 + T_{13}V_3 - U_1 < 0$ mais $V_2 = 1$ et $V_3 = 0$. Donc l'inégalité ci-dessus devient $T_{12} - U_1 < 0$. De la même façon pour V_2, $U_2 < 0$ et pour V_3, $T_{23}-U_3 < 0$.
- Pour l'état stable B, nous avons
  - Pour V_1, $T_{12}+T_{13}-U_1 > 0$
  - Pour V_2, $T_{12}+T_{23}-U_2 > 0$
  - Pour V_3, $T_{23}+T_{13}-U_3 > 0$

Cet ensemble d'inégalités peut être résolu pour attribuer des valeurs aux six inconnues. Nous procédons en choisissant des valeurs arbitraires (comprises entre -1 et 1) pour satisfaire une inégalité à la fois. Nous obtenons alors :

T_{12}	0.5
T_{13}	0.4
T_{23}	0.1
U_1	0.7
U_2	-0.2
U_3	0.4

Tableau C-1

En appliquant ces règles à chaque entrée, le réseau fait correspondre une sortie qui est un état stable le plus proche de l'entrée.

Sans entrer dans les détails, on peut noter que le modèle de Hopfield présente certains problèmes lors de l'apprentissage. **[Aleksander et al. 90]** propose des solutions à ces limitations.

C.3. Modèle multi-couches

Dans cette structure de réseau à couches, chaque couche reçoit des entrées de la couche précédente et transmet ses résultats à la couche suivante. Les couches sont de deux types :

- La ***couche d'entrée*** qui transmet les entrées provenant du monde extérieur à la couche suivante. Il existe une seule couche d'entrée par réseau.
- Les ***couches de traitement*** qui attribuent les poids lors de l'entraînement et calculent les sorties lors de l'exécution du réseau. Ces couches peuvent se décomposer en deux catégories :
  - ◊ La ***couche de sortie*** qui permet au réseau de communiquer ses résultats au monde extérieur.
  - ◊ Les ***couches cachées*** qui n'ont aucun contact avec le monde extérieur.

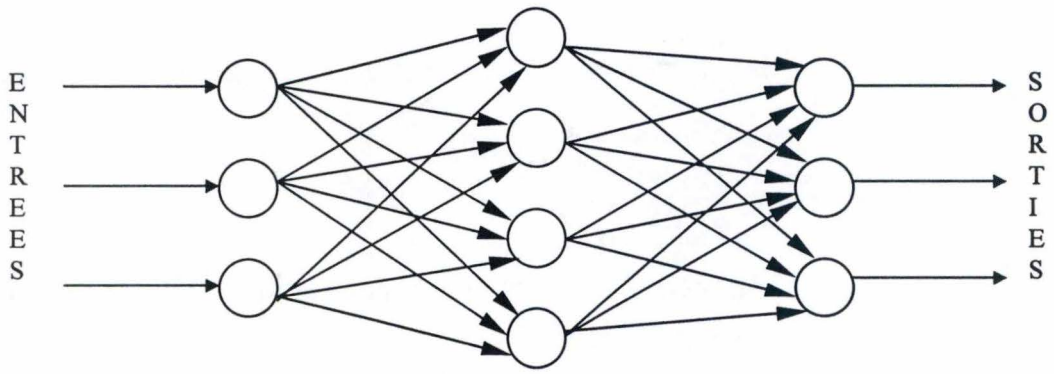


Figure C-6 Réseau multi-couches

Nous allons décrire brièvement un algorithme d'entraînement utilisé avec ce type d'architecture : l'algorithme de *rétro-propagation* (**[Nivelles et al. 90]**). Pour entraîner un réseau de neurones, nous devons ajuster les poids de chaque unité d'une telle façon que l'erreur entre la sortie désirée et la sortie réelle soit la plus faible possible. Ce processus nécessite que le réseau de neurones calcule l'erreur due à l'imprécision des poids attribués aux liaisons. L'algorithme d'apprentissage se présente comme suit :

- Sélectionner l'entrée et la sortie désirée correspondante et la présenter au réseau.
- Calculer la sortie du réseau.
- Calculer l'erreur entre la sortie du réseau et la sortie désirée.
- Ajuster les poids du réseau de façon à minimiser l'erreur.
- Répéter les étapes de 1 à 4 pour chaque entrée de l'ensemble d'apprentissage, et ce jusqu'à ce que l'erreur soit suffisamment faible pour tout l'ensemble.

Nous appliquons ces calculs à l'ensemble des couches du réseau. Il nous faut détailler les opérations qui ont lieu lors du calcul de la sortie d'une cellule et lors de l'ajustement des poids du réseau (**[Nivelles et al. 90]**).

La sortie de chaque neurone est déterminée en faisant la somme des entrées multipliées par les poids des entrées. Cette somme est ensuite

transformée à l'aide de la fonction d'activation, ce qui permet de calculer la valeur de sortie d'une cellule. L'ensemble des valeurs de sortie des cellules d'une couche constituent l'ensemble des entrées de la couche supérieure. D'une manière plus formelle, nous donnons les différentes équations utilisées. Remarquons que nous nous bornons à citer les équations sans les expliquer en détail. La fonction d'activation d'une cellule est définie comme suit :

Equation C-1

$$S = \left(\frac{1}{1 + e^{-Sres}} \right) = g(Sres)$$

où Sres équivaut à la somme des entrées (x_i) multipliées par les poids des entrées (w_i) et noté comme suit :

Equation C-2

$$Sres = \sum_{i=1}^{n} x_i w_i$$

La fonction d'activation est dérivable ce qui donne :

Equation C-3

$$g' = S(1 - S)$$

Pour ajuster les poids du réseau, nous distinguons deux étapes :

- ***Ajuster les poids de la couche de sortie*** : on calcule la différence entre la sortie désirée et la sortie produite. Cette erreur est ensuite multipliée par la dérivée de la fonction d'activation calculée pour cette couche. On obtient un Ψ pour chaque cellule (Sd désignant la sortie désirée) :

  Equation C-4

  $$\Psi = S(1 - S)(Sd - S)$$

  De manière plus pratique, prenons une cellule *c* de la couche de sortie. Pour chaque entrée de cette cellule, on multiplie Ψ par la sortie de la cellule de la couche inférieure. Ce produit est multiplié par un taux d'apprentissage η (choisi entre 0,01 et 0,1) et ce résultat est ajouté au poids de la cellule *c* pour cette entrée.

- ***Ajuster les poids des couches cachées*** : les calculs sont équivalents à ceux utilisés lors du calcul des poids de la couche de sortie sauf pour le calcul de Ψ car les couches cachées n'ont pas connaissance de la sortie désirée. Pour cela, on va *rétro-propager*

l'erreur de la couche de sortie vers la couche d'entrée. Prenons une cellule *c* de la couche cachée. Cette cellule est reliée par des arcs à la couche supérieure. Prenons une cellule de cette couche supérieure que nous notons *s*. Chacun des poids associés aux arcs (de la cellule *c*) est multiplié par le Ψ de la cellule *s* (de la couche supérieure). Pour obtenir la valeur Ψ de la cellule *c*, on calcule la somme de ces produits.

C.4. Réseau de neurones et langage naturel

Une des applications possibles des réseaux de neurones au domaine du langage naturel est la prédiction de la catégorie lexicale du mot suivant (**[Morgan et al. 91]**). Les expériences menées par Nakamura et Shikano (**[Nakamura et al. 89]**) utilisent un corpus découpé en 500 blocs de plus ou moins 2000 mots chacun. Ils « taggent » le corpus, c'est à dire que chaque mot des blocs est remplacé par sa catégorie lexicale. Pour le corpus utilisé (*Brown Corpus English Text Database*), 88 catégories différentes (« tags ») ont été répertoriées. Ensuite, ils créent le réseau de neurones comme suit :

- Une couche de sortie comprenant 89 noeuds (un pour chaque catégorie lexicale et un pour le blanc).
- Deux couches cachées comprenant chacune 16 noeuds.
- Une couche d'entrée comprenant 89 noeuds.

Chaque noeud de la couche d'entrée et de la couche de sortie se réfère à une catégorie lexicale ou a un blanc. Durant l'entraînement, l'entrée et le noeud de la catégorie désirée sont positionnées à un, les autres noeuds de la couche d'entrée et de la couche de sortie sont à zéro. Lors de l'exécution, les valeurs de la couche de sortie sont les prédictions de la catégorie lexicale du mot suivant. Les résultats de ces expériences montrent un pourcentage de 90% de prédiction correcte pour 32000 exemples d'entraînement (**[Morgan et al. 91]**). Une expérience similaire a été tentée par Christopher Vogt avec le corpus *Wall Street Journal*

[Benello et al. 89] ont également développé un réseau de neurones pour lever l'ambiguïté parmi les catégories fournies pour un seul mot. Les mots ambigus ont des définitions et des usages multiples, et par conséquent, ils ont des « tags » multiples. Ils ont repris le corpus « taggé » par Nakamura et Shikano mais ils ont ajouté pour chaque mot l'ensemble des catégories lexicales que celui-ci peut avoir. Le réseau a été entraîné pour sélectionner la catégorie lexicale correcte étant donné les quatre mots précédents, le mot

ambigu et le mot suivant. Ils ont utilisé un réseau à 3 couches se présentant comme suit :

- Une couche de sortie comprenant 88 noeuds (un par catégorie syntaxique).
- Une couche cachée et une couche d'entrée comprenant chacune 236 noeuds. Les quatre mots précédents ont été encodés en utilisant les quinze catégories de Hanson non détaillées ici (**[Hanson87]**). Les deux autres mots sont codés selon les 88 catégories répertoriées dans le corpus.

Durant l'entraînement, la catégorie correcte de sortie est positionnée à 0,5 et les autres à -0,5. Lors de l'exécution, les valeurs de la couche de sortie sont les prédictions quant à la catégorie lexicale correcte du mot ambigu. Les résultats de ces expériences montrent un pourcentage de 95% de prédiction correcte pour 100 phrases de test et avec une entraînement de 900 phrases (**[Morgan et al. 91]**).

Le chapitre IV. présente des possibilités d'utilisation d'un tel système en vue d'une amélioration future de l'application.

Chapitre III
« Le test de Turing » et le Loebner Prize

A. Une architecture d'application

A.1. Introduction

L'objectif qui nous a été attribué était la conception d'une application qui participerait à la compétition du Loebner Prize deux mois plus tard. Une première version de notre application devait être terminée en deux semaines, car une sélection préalable de six participants était prévue par le règlement. Ce manque de temps nous a empêché d'utiliser des techniques élaborées (utilisation de grammaires, analyse syntaxique des phrases, ...) Après notre sélection, nous avons continué à améliorer la première version.

Le « Loebner Prize » est la première compétition relative au *Test de Turing*. Pour rappel, Alan Turing était un brillant mathématicien britannique préoccupé par la question : « Une machine peut-elle penser ? ». Son article paru dans la revue Mind en 1950 présentait le « jeu de l'imitation » où un interrogateur reçoit des réponses d'un ordinateur et d'un être humain. La suggestion de Turing était, que si on ne pouvait faire la

différence entre les réponses de l'ordinateur et les réponses formulées par l'homme, alors l'ordinateur pense.

En 1991, un accord a été pris entre Hugh Loebner et *The Cambridge Center for Behavioral Studies* pour organiser une compétition dans le but d'implémenter le *Test de Turing*. La compétition[1] à laquelle nous avons participé a eu lieu à Sydney en janvier 1998. Le déroulement de la compétition se passe comme suit : dix juges vont devoir départager dix machines (en fonction de leur ressemblance avec des réponses qui auraient pu être formulées par un humain). Les applications des participants, qui sont au nombre de six, s'exécutent dans la **pièce A** et affichent les commentaires produits dans la **pièce B**. Étant donné, qu'il y a dix machines à départager et six participants, quatre personnes (agents) sont également présentes dans la pièce A et formulent des commentaires vers la pièce B de manière tout à fait naturelle. Les juges, qui se trouvent dans la pièce B, n'ont pas connaissance de la provenance des commentaires (personnes ou programmes). Leur fonction est d'attribuer un score qui reflète le caractère humain ou non-humain du terminal leur permettant de dialoguer. Les personnes (juges et agents) ont été choisies de manière aléatoire sans tenir compte de leur expertise dans le domaine de l'intelligence artificielle (journalistes, enseignants, psychologues, ...).

Notre application de type « Eliza » (**[Weizenbaum81]**) donne la possibilité à des personnes (juges) de *converser* avec l'application elle-même. Voici un exemple de conversation que notre application devrait pouvoir réaliser :

JUDGE Do you think that the
JUDGE Republicans can succeed
JUDGE In winning the White House?
PROGRAM Only if Newt succeeds in
PROGRAM developing a more tolerant image

De manière brève, la technique que nous avons choisie pour permettre à notre application de mener une conversation cohérente est relativement simple. Dans une base de données, nous avons rédigé un ensemble de commentaires pouvant être utilisés comme réponses au message d'un juge. Aux commentaires sont associés deux listes de mots que nous appelons « **sujets pré-définis** » et « **mots-clés non sujets** ». Les

[1] *Règles sont disponibles à l'annexe D.*

sujets pré-définis représentent les thèmes généraux (vacances, Australie, Belgique, cinéma, ...) relatifs aux commentaires associés tandis que les mots-clés non-sujets vont restreindre la portée du ou des commentaires(s) par rapport aux thèmes. Par exemple, si le thème du message du juge se rapporte aux animaux, nous pouvons prévoir un commentaire parlant des animaux d'Australie et un autre parlant des animaux de Belgique. Dès lors, les commentaires auront respectivement les mots-clés suivants : kangourou, koala (pour les animaux Australiens) et sanglier, cerf (pour les animaux Belges). Donc, lorsqu'un juge encode un message, nous recherchons si certains des mots employés dans ce message sont repris dans nos sujets pré-définis. Si tel est le cas, nous sélectionnons un des commentaires qui convient le mieux par rapport à l'ensemble des mots figurant dans le message du juge et les mots-clés non sujets associés aux commentaires. De cette manière, nous espérons que notre commentaire suivra le fil de la conversation entamée avec le juge. Par contre, si les mots repris dans le message du juge ne figurent pas dans la liste de nos sujets pré-définis, nous sélectionnons un commentaire *passe-partout*. Par commentaire *passe-partout*, nous voulons dire un commentaire qui est envisageable comme réponse dans la plupart des cas de figure tel que « W*hy do you think that ?* » ou « *I don't think so* ». Mise à part la génération de commentaires comme nous venons de le décrire brièvement, notre application comprend également d'autres fonctionnalités utiles pour rendre notre application la plus « humaine » possible : correction orthographique, simulation de la frappe au clavier, apprentissage.... Ces techniques, tout comme la génération de commentaires, seront reprises en détail dans le paragraphe consacré à l'architecture générale de l'application.

L'implémentation de l'application a été réalisée en Prolog (Sicstus Prolog 3.6) sous environnement Unix. Les bases de données utilisées ainsi que les principales fonctions utilisées seront présentées dans ce chapitre.

A.2. Architecture principale

En plus de la réalisation de l'application, nous avions d'autres fonctionnalités à implémenter telles que l'initialisation de l'application lors de l'arrivée d'un nouveau juge et transcription de la conversation dans un fichier tout en respectant le standard prévu par le superviseur de la

compétition. Ces fonctionnalités avaient pour but de respecter une certaine cohérence parmi les différentes applications participant à la compétition.

Dans ce paragraphe, nous allons décrire de façon détaillée l'architecture employée pour l'élaboration de notre application (voir figures III-1 et III-2). Celle-ci se compose des modules relatifs à la gestion de la conversation avec les juges (déjà implémentés ou à implémenter dans le futur), des modules demandés pour maintenir une certaine cohérence parmi les applications et les bases de données.

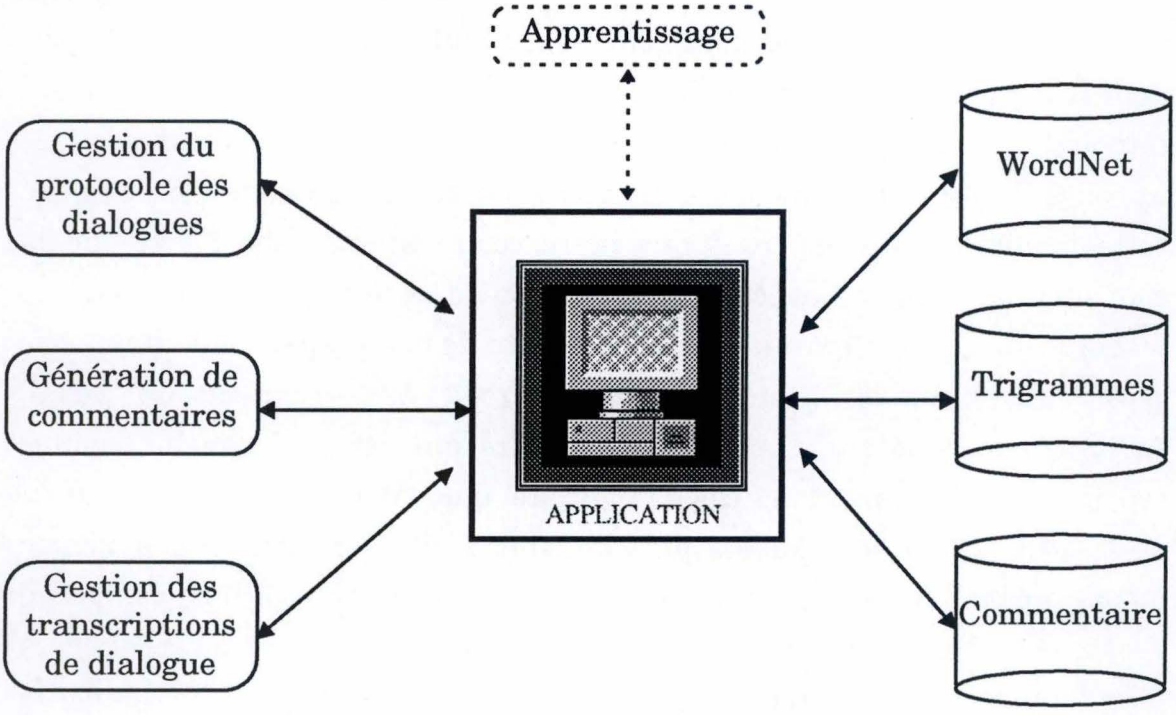


Figure III-1 Architecture de l'application

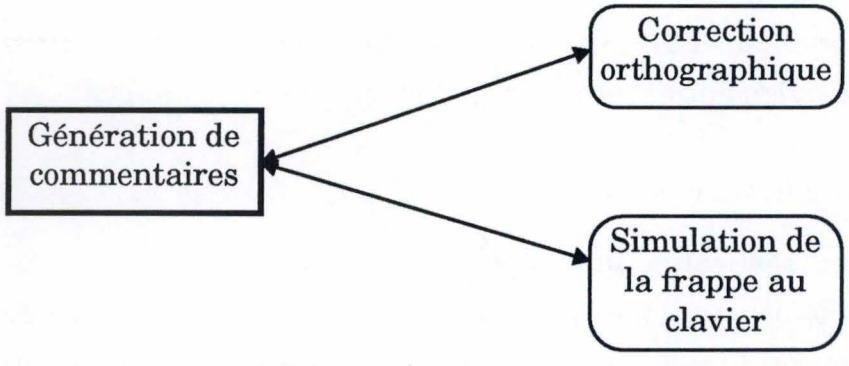


Figure III-2 Interactions du module *Génération de commentaires*

Avant de décrire les différents modules en détail, ceux-ci seront définis de manière brève pour en faciliter la compréhension. Les modules de notre architecture sont les suivants :

- ***Gestion du protocole des dialogues*** : module qui permet de préparer le programme au départ de la compétition et de gérer les interventions des différents juges. Ce module était imposé par le règlement de la compétition.

- ***Génération de commentaires*** : module principal qui analyse la phrase et produit un commentaire relatif à cette phrase. Ce module fait également appel à deux autres modules qui sont *Correction orthographique* et *Simulation de la frappe au clavier*.

- ***Gestion des transcriptions de dialogue*** : module qui permet de transcrire les dialogues dans un fichier accompagnés d'autres informations. Ce module était également imposé par le règlement de la compétition.

- ***Correction orthographique*** : module qui permet de corriger certaines erreurs que la personne (le juge) aurait introduit au terminal.

- ***Simulation de la frappe au clavier*** : module qui permet d'imiter la vitesse de frappe et les erreurs commises par un humain lorsque celui-ci encode des données à l'aide du clavier.

- ***Apprentissage*** : module non implémenté par manque de temps qui aurait permis au programme d'acquérir au fur et à mesure les différentes structures de phrases encodées par le juge et donc, d'être capable de formuler ces propres phrases. Les différentes techniques vues dans le chapitre précédent (voir *Apprentissage par les machines*) pourront être appliquées et combinées pour implémenter ce module.

En ce qui concerne les bases de données, nous nous contentons ici de présenter leurs implémentations physiques. Les différentes utilisations de ces bases de données seront reprises au cours des paragraphes suivants :

- ***Wordnet*** : ensemble de mots anglais ainsi que leurs synonymes et hypernymes. Ce module sera détaillé ultérieurement (voir la paragraphe consacré à *WordNet*).

- ***Trigrammes*** : ensemble de triplets de caractères que nous avons créés à partir du corpus « Alice in Wonderland ». Chaque trigramme est représenté par un fait Prolog. Ces données vont nous être utiles lors de la correction orthographique.

- ***Commentaires*** : ensemble des sujets pré-définis et des commentaires préparés comme éventuelles réponses à une formulation d'une personne (le juge). Les sujets pré-définis ainsi que les commentaires sont représentés par des faits Prolog. Les sujets pré-définis sont représentés comme suit :

```
sub(belgium,belgium).
sub(namur,belgium).
sub(liege,belgium).
```

Le premier argument est un mot qui peut se lier à un thème général (ou sujet pré-défini) et le second argument est le sujet pré-défini. Pour les commentaires, six types de faits se présentent :

```
gener_com_quest(j7,_,[animal],[forest],'In Belgium we have some nice
            animals in forest like deers or wild boars.').
Description : fait utilisé pour donner une réponse à question
Arguments : le numéro identifiant du fait
            le mot interogatif[2] du message encodé par le juge
            la liste des sujets pré-définis pour ce commentaire
            la liste des mots-clés non sujets de ce commentaire
            le commentaire

gener_com_aff(e15,[school],[],'My studies are very interesting.').
Description : fait utilisé pour donner une réponse à une affirmation
Arguments : le numéro identifiant du fait
            la liste des sujets pré-définis pour ce commentaire
            la liste des mots-clés non sujets de ce commentaire
            le commentaire

predef([thank,you|Suite],'No problem.').
Description : fait utilisé pour donner une réponse à une phrase
            pré-définie[3]
Arguments : la liste des mots commencant le message
            la réponse brève formulée pour ce message
```

[2] *Ce mot interrogatif peut nous permettre de répondre encore plus précisément à la question.*
[3] *Une phrase pré-définie est un type de message fréquent tel que « Hello. », « Thank you. » ou « Good night. » et pour lesquels nous avons prévu de brèves réponses.*

```
red_quest(holiday,_,'Can you go on holiday this year?').
Description : fait utilisé pour afficher une question[4] sur le sujet
              (thème) introduit par le juge.
Arguments :  le sujet abordé par le juge
             le terme indiquant si une pause est faite après
             l'affichage de cette question
             la question formulée sur ce sujet

gener_com_sec(2,'Can you talk more about that...?').
Description : fait utilisé pour donner une réponse lorsque nous
              n'avons pas trouvé de sujet dans l'affirmation encodée par
              le juge
Arguments :  le numéro identifiant du fait
             la réponse brève formulée pour ce message

gener_sec(6,'I believe that you asked enough questions for today.').
Description : fait utilisé pour donner une réponse lorsque nous
              n'avons pas trouvé de sujet dans la question encodée par le
              juge
Arguments :  le numéro identifiant du fait
             la réponse brève formulée pour ce message
```

a) Gestion du protocole des dialogues

Le module de gestion du protocole des dialogues était exigé de tous les participants de la compétition. Ceci permettait de garder une cohérence entre les différents programmes mais également d'éviter aux juges de mémoriser différentes façons de sortir du programme, changer de juge, ... Ce module s'exécute selon deux modes qui sont :

- Le mode *initialisation*
- Le mode *compétition*

(1) Le mode *Initialisation*

Lors du début de la compétition, le mode *initialisation* permettra au superviseur de la compétition de préparer le programme. Ces différentes tâches peuvent avoir lieu durant l'initialisation :

- Demande du nom de fichier dans lequel les conversations entre le terminal et le juge seront transcrites.

- Conversation entre le superviseur (appelé judge00 dans le fichier des transcriptions) et la machine. Durant cette conversation, le prompt à l'écran reste « + ». Celui-ci changera lorsque le superviseur encodera « @@T [CR][CR][5] ».

---

[4] *L'utilisation de ce commentaire sera détaillé plus tard sous le terme de commentaire de* continuité.
[5] *[CR] représentant un passage à la ligne (Carriage Return).*

- Inscription d'un en-tête dans le fichier des transcriptions :

(c)1998 Cambridge Center For Behavorioral Studies all rights reserved.
[ELISABETH] - [REALIZED BY V.BASTIN AND D.CORDIER]
Start at: [1998/01/11 09:55:38]

(2) Le mode *compétition*

Durant la compétition, le juge a la possibilité d'encoder certaines séquences particulières dans la perspective d'un traitement spécial, qui dépendra également du *prompt* se trouvant à l'écran à ce moment. L'ensemble des séquences possibles sont présentées comme suit :

- « @@T [CR][CR] » : à la suite d'un prompt « > » ou « + », cette séquence indique la terminaison de la conversation avec le juge actuel. L'écran sera également réinitialisé avec un prompt « ? » d'attente du juge suivant.

- « @@nn [CR][CR] » : à la suite d'un prompt « > » ou « ? », cette séquence indique l'arrivée d'un nouveau juge (le juge nn). Remarquons que chaque juge doit s'identifier lui-même avant de débuter la conversation.

- « @@X [CR][CR] » : à la suite d'un prompt « > » ou « ? », cette séquence indique la fin de la compétition et donc la fin de l'exécution du programme.

Il est également important de signaler que lors de la compétition, certaines propriétés doivent être vérifiées à tout instant telles que :

- Le programme doit afficher un commentaire initial à l'arrivée de chaque juge.
- Le commentaire du programme peut être multi-lignes.
- Le prompt « > » signifiant que le juge peut encoder son prochain message ne sera affiché que lorsque le programme aura terminé l'affichage de son commentaire.
- Les questions ou commentaires du juge suivent le prompt « > » et peuvent être multi-lignes. Chaque ligne sera terminée par un « [CR] ».
- Le commentaire du juge, lorsque celui-ci a le prompt « > », sera affiché caractère par caractère à l'écran.

- L'encodage de « [CR][CR] » indiquera que la question ou le commentaire du juge est complet et qu'il attend une réaction de son interlocuteur.
- Les caractères affichés à l'écran seront de taille égale.

(3) Exemple

Voici l'exemple d'un programme répondant à une interaction avec le juge numéro 4.

>@@04[CR]	(Arrivée d'un nouveau juge, juge 4)
>[CR]	(Deux passage à la ligne requis)
Welcome judge 4	(Commentaire du program)
> Do you think that the [CR]	(Question multi-ligne)
> Republicans can succeed [CR]	
> in winning the White House? [CR]	(Premier passage à la ligne)
> [CR]	(Second passage à la ligne)
Only if Newt succeeds in	(Réponse multi-ligne du programme)
developing a more tolerant image.	
>_	(Prompt ">", le programme attend le commentaire du juge)
^ Curseur attend l'encodage du juge	

b) Génération de commentaires

Avant de débuter la description de ce module, nous croyons qu'il serait intéressant de définir la terminologie pour éviter toute confusion par la suite. Voici le détail de cette terminologie :

- ***Mots-clés*** : ensemble de mots de la phrase qui nous permettra de procéder à une recherche de sujets.
- ***Sujets pré-définis*** : ensemble de sujets (noms communs ou thèmes) pour lesquels nous avions défini des commentaires.
- ***Mots-clés sujets*** : ensemble de mots de la phrase qui figurent parmi nos sujets pré-définis.
- ***Mots-clés non sujets*** : ensemble de mots qui ne figurent pas parmi les sujets mais qui peuvent nous servir par la suite pour sélectionner le commentaire le plus adéquat au message du juge.
- ***Sujet-verbe pré-défini*** : ensemble de sujets (verbes) pour lesquels nous avions défini des commentaires.

Dès le départ, la liste de nos sujets pré-définis et nos commentaires sont stockés dans la base de données *Commentaires* (voir descriptif de cette base de données). Pour bien comprendre le fonctionnement de ce module, il est utile de visualiser le schéma d'enchaînement des traitements (figure III-3), que nous décrirons par la suite.

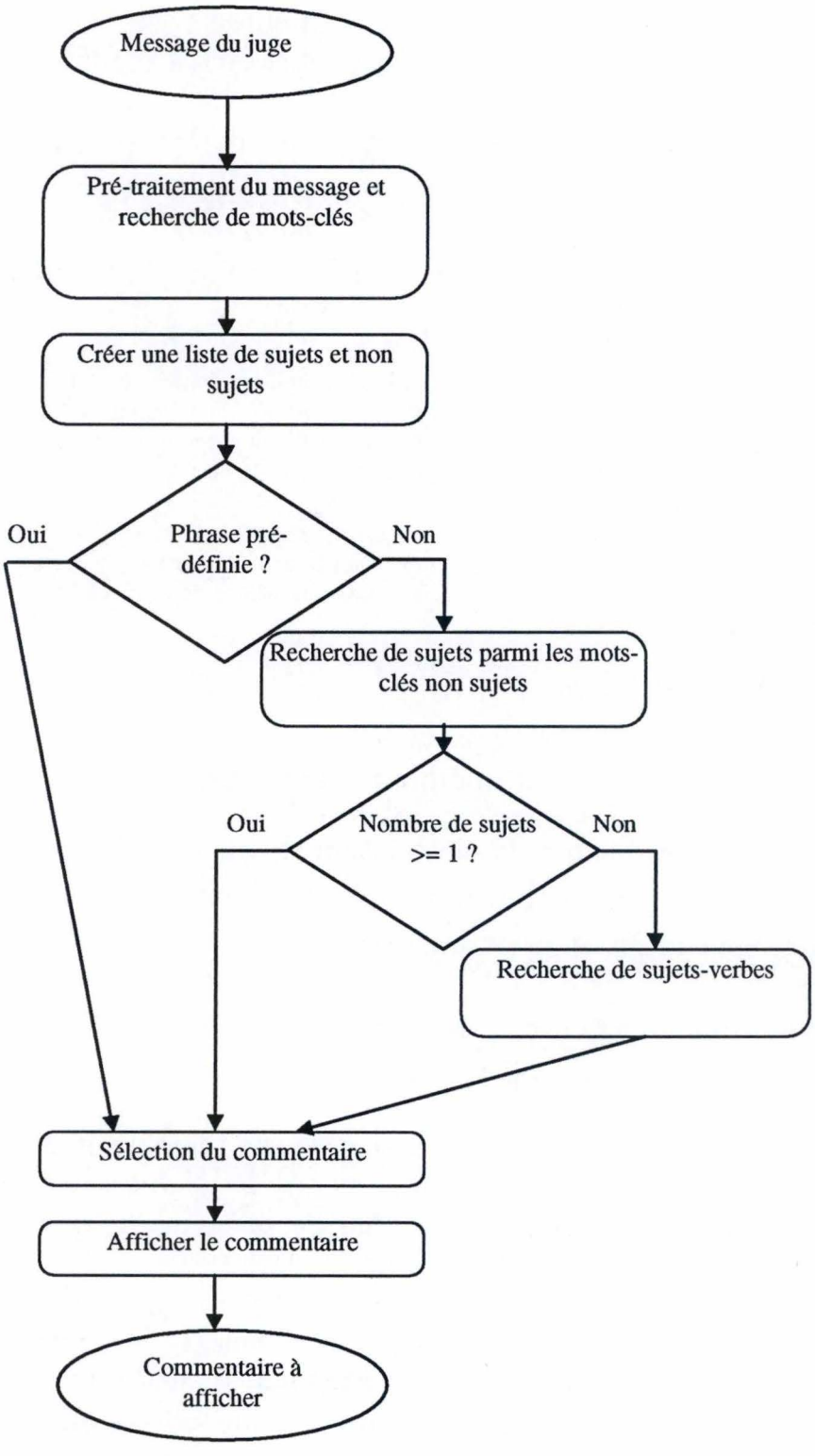


Figure III-3 Génération d'un commentaire

(1) Pré-traitement du message et recherche de mots-clés

Lorsque le message de l'utilisateur nous parvient, nous devons lui faire subir une certain nombre de modifications pour permettre les différents traitements ultérieurs. Les modifications de la chaîne initiale seront les suivantes :

- Transformer les caractères en minuscules.
- Enlever les caractères de ponctuation.
- Extraire les mots (grâce au séparateur « blanc ») qui vont nous être particulièrement utiles lorsque nous devrons procéder à la recherche de mots-clés.

Ensuite, nous allons analyser le message encodé. Nous entendons par là que nous allons extraire tous les mots-clés dans le message de manière à ce que ceux-ci puissent nous être utile lors de la formulation d'une réponse. Ces mots-clés reprennent l'ensemble des substantifs du message. Pour cela, nous avons créé une liste de faits PROLOG comprenant des mots non-substantifs (pronoms, prépositions et adjectifs) les plus fréquents. Lors de l'analyse d'un mot (repris dans le message du juge), nous comparons celui-ci à notre liste de faits. Si le mot est absent de cette liste, nous le considérons comme mot-clé.

(2) Créer une liste de sujets et non sujets

Lorsque nous avons les mots-clés à notre disposition, nous pouvons commencer à traiter ceux-ci. Ce traitement nous permet de répertorier les mots-clés qui figurent dans notre liste de sujets pré-définis. De manière pratique, nous parcourons les faits de notre base de données *Commentaires* à la recherche d'un fait Prolog dont le premier argument serait un de nos mots-clés. Lorsqu'un fait Prolog est trouvé, le sujet pré-défini (second argument) est ajouté à la liste de sujets que nous sommes en train de créer. Si aucun fait Prolog n'est trouvé, ce mot-clé est inséré dans la liste de non sujets car ceux-ci seront utiles lors de la *sélection du commentaire à afficher* (voir ci-après).

(3) Phrase pré-définie ?

Tout d'abord, définissons ce qu'est une phrase pré-définie. Dans notre fichier, en plus des commentaires, nous avons prévu un ensemble de phrases relativement courantes qui pourraient être utilisées comme message par le juge. Ce sont des phrases pré-définies comme par exemple : « Good night. », « How are you ? », « Nice to meet you. », « Bye. », « Hello. », ...

Si le juge a bel et bien encodé une phrase pré-définie, nous pouvons immédiatement sélectionner un commentaire se trouvant dans la base de données *Commentaires* tel que : « No problem. », « Very fine, thanks. », « We have the same opinion. », « Ok. », ... Dès lors, pourquoi créer une liste de mots-clés sujets et de non-sujets ? Car lorsque nous affichons le commentaire, nous avons également la possibilité d'afficher un commentaire *passe-partout* (voir *Sélection du commentaire à afficher*) qui sera choisi sur base des mots-clés sujets et mots-clés non-sujets pour respecter le contexte du message encodé par le juge.

Si ce n'est pas une phrase pré-définie, nous continuons les étapes suivantes à la recherche d'autres mots-clés sujets.

(4) Recherche de sujets parmi les mots-clés non-sujets

La technique que nous utilisons est d'avoir le plus de sujets à notre disposition pour bien cerner le contexte du message et répondre à celui-ci de la manière la plus cohérente possible. Pour ce faire, nous lançons un second traitement sur les mots-clés non sujets. Cette recherche utilise la base de données *WordNet*. Tout d'abord, nous allons chercher le numéro d'identifiant de ce mot dans les faits Prolog (***s(identifiant,_,libellé du mot-clé,_,_)***). Pour trouver des synonymes, nous recherchons dans ces faits (***s(id,_,lib,_,_)***), ceux qui ont le même identifiant et un libellé différent. A ce stade, nous connaissons les libellés de synonymes trouvés. Pour trouver des hypernymes, nous recherchons dans d'autres faits (***hyp(id,identifiant de l'hypernyme)***). Cette recherche nous rend les numéros d'identifiants des hypernymes et va s'exécuter de manière récursive[6] en remontant

[6] *On recherchera également l'hypernyme de l'hypernyme du mot-clé et ainsi de suite.*

jusqu'à quatre niveaux de hiérarchie. Enfin, nous parcourons les faits (***s(id,_,lib,_,_)***) pour connaître les libellés des hypernymes trouvés.

Pour des raisons de performance, nous limitons cette recherche à trois mots-clés non sujets et nous l'exécutons en parallèle avec un autre processus d'affichage d'un commentaire *d'attente*[7] comme : « I think… », « Wait a minute. », « Oh, oh, … », « That's interesting. ».

Mot-clé non sujet	Synonymes	Hypernymes	Sujets
koala	*dog, domestic dog, canine, …*	*vertebrate, animal, …*	animal
cinema	*film, movie, …*	*culture, building, …*	movie, culture
hamburger	*sandwich, beefburger, …*	*food, nutriment, dish, …*	food

Tableau 1- Recherche de synonymes et hypernymes

(5) Nombre de sujets >= 1 ?

Lors de ce test, nous vérifions si nous avons trouvé des sujets. Deux cas sont possibles : oui et non.

Si nous n'avons pas de sujets à notre disposition, nous prenons la suite des mots-clés non sujets qui n'avaient pas été pris en compte lors de l'étape *Recherche de sujets parmi les mots-clés non sujets* et nous exécutons le même traitement sur ceux-ci.

Si des sujets ont été trouvés, nous passons à l'étape suivante qui est : *Sélection du commentaire*.

(6) Recherche de sujets-verbes

C'est le même principe que lors de la recherche de sujets mais nous l'effectuons avec des verbes. Ces sujets-verbes que nous pouvons trouver vont nous aider à mieux cerner le contexte de la phrase. Dans la suite de ce texte, le terme mots-clés sujets fera référence à des noms ou verbes qui étaient présents dans notre liste de sujets pré-définis.

(7) Sélection du commentaire à afficher

Lors de cette étape, nous avons imaginé une petite astuce implémentée facilement grâce au Prolog. Lorsque le juge encode un

message, il peut très bien essayer de tromper l'ordinateur en encodant plusieurs fois le même message l'un à la suite de l'autre ou des messages différents comprenant les mêmes sujets. Pour remédier à cela, nous avons préparé plusieurs commentaires se rapportant aux mêmes sujets. Ces commentaires sont implémentés sous forme de faits Prolog. Pour rappel, Prolog charge les faits en mémoire dès le chargement du programme et rend le premier fait Prolog qui est adéquat avec nos sujets et mots-clés non-sujets. C'est à partir de là que nous avons imaginé une technique que nous nommerons *carrousel*. Cette technique fonctionne comme suit : lorsque un fait a été sélectionné, celui-ci est retiré de la mémoire et ensuite ajouté a la fin de la liste des autres faits en mémoire. De la sorte, ce fait ne sera pas repris de suite si le juge essaie de nous tromper.

De plus, si nous n'avons pas trouvé de sujets lors de cette étape, nous faisons appel au module *Correction orthographique* (voir le paragraphe 2.c). Ce module permet d'avoir accès à de nouveaux mots-clés qui auraient été mal orthographiés par le juge. Une recherche de sujets est ensuite opérée sur ces nouveaux mots-clés. En fonction de sujets trouvés ou non, trois possibilités de sélection d'un commentaire se présentent à nous, à savoir :

- Le message encodé par le juge est une phrase pré-définie (voir *Phrase pré-définie*). Dans ce cas, nous sélectionnons un des commentaires déjà rédigés et nous exécutons la technique du *carrousel.*

- Le message encodé n'est pas une phrase pré-définie et comprend un ou plusieurs sujets. Dans ce cas, nous allons rechercher l'ensemble des commentaires[8] dans lesquels figurent au moins un mot-clé sujet. Ensuite nous recherchons dans cet ensemble de commentaires, ceux qui conviennent le mieux avec nos mots-clés non sujets. De cette manière, nous ne retenons qu'un ensemble de commentaires cohérents avec les thèmes cités (mots-clés sujets et mots-clés non sujets) dans le message du juge (exemple dans le tableau ci-dessous). Enfin, nous sélectionnons le premier de ces commentaires et nous exécutons la technique du *carrousel.*

---

[7] *Ce commentaire nous permet simplement de masquer le temps prolongé que pourrait prendre la recherche de synonymes et hypernymes.*

Mot-clé sujet	Mot-clé non sujet	Commentaire
Animal	*snake, spider*	I like animals but not snakes and spiders.
Animal	*lion, elephant*	I don't like wild animals.

Tableau 2

En plus de cela, nous générons également un commentaire de *continuité*. Nous appelons commentaire de *continuité*, un commentaire qui a pour but d'approfondir le sujet choisi par le juge en posant des questions sur le sujet. Par exemple, si le juge entame une conversation sur les vacances, nous afficherons des commentaires de *continuité* comme : « Where did you go on holidays last year? », « Do you like the beach ? ». De plus, à la fin de l'affichage du commentaire spécial « *Sorry, I'm going to toilets.* », notre application marque une pause de quelques secondes. Ceci pour donner un peu plus de caractère humain à notre application et semer le doute dans l'esprit du juge. Remarquons que l'affichage du commentaire de *continuité* n'a pas lieu à chaque message du juge (existence aléatoire) et que la technique du *carrousel* est également d'application parmi ces commentaires de *continuité*.

- Le message encodé n'est pas une phrase pré-définie et ne comprend pas de sujets. Dans ce cas, nous utilisons une technique de réécriture du message en inversant la phrase. Par exemple, si le juge encode le message « Do you like pizza ? » et que le programme ne trouve aucun sujet, alors celui-ci répond « Do I like pizza ? ». Ce principe est le même pour les affirmations. En plus de ce commentaire, nous sélectionnons également un commentaire *passe-partout*[9] comme « Why do you think that ? », « I believe that you asked enough questions for today. ».

(8) Afficher le commentaire

Une fois le ou les commentaire(s) sélectionné(s), nous affichons celui-ci en faisant appel à la routine *Simulation de la frappe au clavier*.

[8] *Un commentaire est représenté par un fait Prolog qui contient la liste des mots-clés sujets et des mots-clés non-sujets compatibles avec ce commentaire.*

[9] *Ce commentaire va nous permettre de masquer notre manque de réponses.*

c) Correction orthographique

(1) Introduction

Comme on l'a déjà indiqué, une partie de notre temps a été consacrée à l'implémentation d'un système de correction orthographique, que l'on devait ajouter comme module au programme de type « Eliza » déjà réalisé. Nous étions totalement libres sur le choix de la méthode. Cependant, certaines contraintes ont contribué au choix du langage (Prolog) et aux méthodes utilisées. Voici les principales contraintes dont nous avons dû tenir compte :

- Etant donné que les modules de correction orthographique devaient être insérés dans le premier programme que nous avions réalisé, il était alors plus simple d'écrire le module dans le même langage : le PROLOG.
- La place disponible sur le disque était fortement limitée. En effet, il nous restait à peine plus de 12 Mb. Il fallait donc prendre cela en considération lors de la construction des bases de données.
- La vitesse devait être acceptable, puisque le programme dans lequel le module de correction orthographique allait être inséré devait imiter le comportement d'un humain tapant au clavier. Il ne fallait donc pas dépasser quelques secondes pour accéder aux bases de données et envoyer les résultats.
- L'utilisateur ne pouvait pas intervenir dans la correction (par exemple en choisissant le mot le plus approprié remplaçant un mot inconnu...).

(2) Les méthodes employées

Les méthodes de correction orthographiques sont très nombreuses, et nous avons trouvé très peu de bibliographie en ce qui les concerne. Nous nous sommes basés sur un article de **[Powers97]** qui reprenait les erreurs de substitution de mots les plus fréquentes. Nous avons traité un type d'erreur de substitution, celui de la *proximité de touches*. Cette erreur vient du fait que les utilisateurs ont parfois tendance à taper sur une touche adjacente à celle qu'ils veulent réellement utiliser.

Les chercheurs du département dans lequel nous travaillions nous ont conseillé de vérifier le fait que les utilisateurs ne doublent pas une lettre en tapant trop vite, ou au contraire n'oublient pas de doubler une lettre. Nous avons implémenté cette caractéristique également.

Au cas où aucune de ces méthodes ne fonctionne, nous avons ajouté un module qui essaie de calculer les *distances* reflétant la différence syntaxique entre un mot inconnu (non trouvé dans le dictionnaire) et des mots présents dans la base de données. On choisi alors le mot dont la distance le séparant du mot cible est la plus faible.

Un inconvénient évident de ces méthodes, auquel il est possible de remédier dans le futur, est le fait que la correction est uniquement syntaxique. On ne tient pas compte de la « distance sémantique » entre les mots, ni du contexte.

(3) Réalisation des modules de correction orthographique

Nous décrivons ici de manière informelle le fonctionnement des différents modules de correction orthographique, dans l'ordre où ils sont appliqués lorsqu'on rencontre des mots inconnus.

(a) Proximité des touches

Ce module est assez simple. Chaque fois qu'un mot n'est pas repris dans le dictionnaire, on essaye de générer tous les mots possibles en changeant chaque lettre du mot par les lettres adjacentes sur le clavier. Pour chaque nouveau mot ainsi généré, on vérifie s'il se trouve dans le dictionnaire. On arrête le traitement dès qu'un mot généré est trouvé dans le dictionnaire.

Pour permettre ce traitement, nous avons réalisé une petite base de données reprenant simplement chaque touche du clavier, suivie par toutes les touches adjacentes.

Voici un extrait de ce fichier :

```
a,z,s,q,_,_,_
f,g,v,c,d,r,t
...
```

Par exemple, en supposant qu'un utilisateur introduise le mot *hohse*, le module va générer la séquence de mots suivante :

```
goyse
yoyse
...
house
```

Il faut être attentif au temps que met le système pour déterminer les possibilités dans le cas de très long mots.

(b) Doublement / manquement d'une lettre

Ce module est utilisé si aucun mot n'a été trouvé par le module précédent. Le fonctionnement de ce module est séparé en deux parties. La première consiste à générer tous les mots en doublant chaque lettre (sauf la première) et vérifiant si le nouveau mot est présent dans le dictionnaire. La seconde partie du traitement consiste à générer des mots en supprimant les doublons de lettres un à un et en vérifiant si un mot généré est présent au dictionnaire. Le traitement s'arrête dès qu'un mot généré est présent dans le dictionnaire.

(c) Calcul de distance entre mots

Ce module génère un masque de recherche destiné à déterminer un ensemble de mots qui remplissent certains critères. Pour cela, nous avons créé un fichier contenant des trigrammes, déterminés à partir de l'analyse d'un ensemble de textes (corpus). Nous avons simplement lu le corpus, et enregistré les statistiques de tous les ensembles de trois lettres apparaissant dans celui-ci.

Ainsi la phrase « Anne mange du pain » génère les trigrammes suivants :

Tr.	*Occ.*
_,a,n	1
a,n,n	1
n,n,e	1
n,e,_	1
e,_,m	1
_,m,a	1
m,a,n	1
...	

Dans un premier temps, le fichier créé était un fichier ASCII classique (le fichier était généré par un script Perl). Ensuite, nous avons réalisé un programme en PROLOG relisant le fichier ASCII généré tout en créant un fichier indexé lisible par des prédicats PROLOG. Il faut maintenant encore expliquer comment nous avons utilisé cet ensemble de trigrammes.

Quand le programme détecte un mot inconnu, pour lequel aucune des deux méthodes précédentes n'a fonctionné, il commence par construire une liste de trigrammes non repris dans sa base de données. C'est sur base de cette liste qu'il va construire des masques de recherche adéquats.

La liste consiste simplement en un ensemble de nombres, chacun d'entre eux étant le centre (concept) du trigramme non reconnu. A ce moment, on considère que le concept est correct, et que l'erreur se trouve soit avant (contexte gauche), soit après celui-ci (contexte droit).

Voici un exemple de masque généré lorsque le programme rencontre le mot *hovidays* . En supposant que le trigramme *hov* soit inconnu, le masque généré sera [_,o,_,i,d,a,y,s]. Le programme va alors accéder à la base de données pour retrouver les trigrammes ayant *o* comme concept (par ordre de probabilité d'apparition). Le trigramme *hol* devrait alors être trouvé, et le mot *holidays* pourra être reconstitué.

Un tel système fonctionne bien dans certains cas mais est relativement imprécis. En effet, suivant le corpus lu, le trigramme *hov* peut exister ou non. Si le trigramme existe, l'erreur risque de ne pas être détectée. Inversement, si plusieurs trigrammes n'apparaissent pas dans la base de données, il est difficile de constituer un masque de recherche valable. Ainsi, en supposant que les trigrammes *hov* et *ovi* n'existent pas, le masque généré serait [_,_,_,_,d,a,y,s], ce qui est trop aléatoire. Dans tel cas, nous avons choisi de ne prendre en compte que le premier trigramme inconnu. Par contre, si des trigrammes inconnus ne se chevauchent pas, nous générons le masque adéquat. Si *hov* et *ays* manquent, nous aurons alors [_,o,_,i,d,_,y,_].

Lorsque le programme a la liste des mots correspondant au masque de recherche, il faut alors calculer une distance entre chacun de ces mots et le mot inconnu. Nous avons choisi la distance la plus simple qui soit : le nombre de lettres différant du mot cible.

d) Simulation de la frappe au clavier

Lorsque un humain encode un message au clavier, nous constatons que celui-ci commet des erreurs. Parfois il les corrige. De plus, la vitesse de

frappe n'est pas régulière. Ce traitement va nous permettre de simuler ces faits à l'aide de trois fonctions :

- ***Pour simuler la vitesse de frappe***, nous générons un nombre aléatoire de milli-secondes d'attente entre l'affichage de chaque caractère. La borne supérieure pour choisir ce nombre aléatoire sera plus large lorsque nous arriverons à la fin d'un mot (reconnaissable par un espace). De la sorte, ce délai d'attente plus long permettra de simuler la réflexion d'un humain après avoir encodé un mot.
- ***Pour simuler les erreurs commises*** (nombre d'erreurs est aléatoire) qui peuvent être de deux types :
  - ◊ Proximité clavier : on insère une erreur dans le commentaire qui dépend de la lettre correcte, c'est à dire que la touche-clavier de l'erreur se trouve directement à côté de la touche de la lettre correcte du commentaire. Exemple : les touches de proximité clavier de la lettre « h » sont : « t », « y », « u », « g », « j », « b » et « n ».
  - ◊ Aléatoire : on insère dans le commentaire une lettre tout à fait indépendante du reste du commentaire.

- ***Pour simuler la correction de l'erreur***. Remarquons que le choix de corriger l'erreur ou pas est aléatoire. La correction peut être envisagée de deux manières :
  - ◊ Retour en arrière immédiat : après insertion de l'erreur dans le commentaire, on insère directement à sa suite un *backspace* et la lettre correcte.
  - ◊ Retour en arrière lointain : après insertion de l'erreur dans le commentaire, on insère une série de *backspaces* un nombre aléatoire de lettres plus tard. Ceci pour simuler l'erreur commise par un humain, qui se rend compte de celle-ci beaucoup plus tard.

e) Gestion des transcriptions des dialogues

Comme il était spécifié dans les règles de la compétition, nous avons également produit un fichier des transcriptions, c'est à dire de l'ensemble des interactions avec les juges. En plus des traitements d'initialisation du fichier (voir *Gestion du protocole des dialogues*), nous avons réalisé d'autres traitements sur le fichier durant la compétition tels que :

- Affichage d'un en-tête à chaque arrivée d'un nouveau juge.
- Affichage de l'identification du juge et de l'heure en ce qui concerne un message encodé par le juge.

- Affichage de « PROGRAM » et de l'heure en ce qui concerne une réponse fournie par notre application.

Voici comme exemple une de nos meilleures transcriptions réalisées durant la compétition. C'est la conversation de notre programme avec le juge 7.

*** JUDGE07 ***
PROGRAM[11:20:57]Hello, I'm Elisabeth. How are you?
JUDGE06[11:21:02]G'day Elizabeth
JUDGE06[11:21:02]May I call you Liz?
PROGRAM[11:21:15]Hello, it's a nice day to speak but I will prefer to speak
PROGRAM[11:21:15]with you on the beach
JUDGE06[11:21:18]Liz, which beach do you like?
PROGRAM[11:21:22]I like going to the beach. It's better in Australia
PROGRAM[11:21:22]because the sea is very clean.
PROGRAM[11:21:22]Do you like the beach?
JUDGE06[11:21:30]Yes
JUDGE06[11:21:30]would you like to go to the beach this afternoon with me?
PROGRAM[11:21:33]Sea, sun and sand. It's wonderful.
JUDGE06[11:21:50]yes, but what about our date this afternoon?
PROGRAM[11:21:57]It's difficult to find something to tell about this.
PROGRAM[11:21:57]I adore eating and I like cooking .
JUDGE06[11:22:16] What food do you like best?
PROGRAM[11:22:20]I like eating and I like often going to the restaurant.
JUDGE06[11:22:28]what is your favourite food?
PROGRAM[11:22:35]In Belgium, we are very fond of food.
PROGRAM[11:22:35]Have you a lot of friends?
JUDGE06[11:22:40]yes
JUDGE06[11:22:40]what about you?
PROGRAM[11:22:42]Really?
JUDGE06[11:22:48]yes, it's true!
PROGRAM[11:22:57]That's interesting.
PROGRAM[11:22:57]Can you talk more about that...?
JUDGE06[11:23:16] I like to fly, do you?
PROGRAM[11:23:20] Why do you think that?
JUDGE06[11:23:30] because
PROGRAM[11:23:35] Can you tell me more about that?
JUDGE06[11:23:45] lying a plane gives me freedon^H,^Hm
PROGRAM[11:23:47] I also like this kind of transport but I'm going often
PROGRAM[11:23:47] with my car.
JUDGE06[11:24:16] Liz, what is your native language?
PROGRAM[11:24:20] I just know French but in this test we must speak English.
PROGRAM[11:24:20] Do you know French?
JUDGE06[11:24:36] un petite peu...
JUDGE06[11:24:36] what language would you prefer to use?
PROGRAM[11:24:40] I just know French but in this test we must speak English.
JUDGE06[11:24:52] OK, that's better for me as well
PROGRAM[11:24:57] You speak a lot today.
JUDGE06[11:25:09] Well, we have to because that's how we evaluate your human-ness
PROGRAM[11:25:12] I think...
PROGRAM[11:25:12] It's my business and not yours.
JUDGE06[11:25:25] well, today, it's mine too
PROGRAM[11:25:30] I'm sure that you have a calendar somewhere.
PROGRAM[11:25:30] Sorry, I'm going to toilets
JUDGE06[11:25:46] better be quick, there's another test session starting soon
PROGRAM[11:25:50] That's interesting.

PROGRAM[11:25:50] better be quick there is another test session starting soon I don't know what to tell
PROGRAM[11:25:50] you.
JUDGE06[11:26:00] adieu, cher Liz
PROGRAM[11:26:02] Is it right?
JUDGE06[11:26:56] bye
PROGRAM[11:26:59] Bye, nice to meet you.
JUGE007 [11:27:05s] @@T
*** JUDGE02 ***

B. WordNet[10]

B.1. Introduction

WordNet est un système reprenant un ensemble de mots anglais et indiquant un certain nombre de relations que possèdent ces mots entre eux (synonymes...). Le but des concepteurs du réseau est de modéliser un grand nombre de relations sémantiques entre les mots codés. La structure de WordNet est basée sur des résultats de recherche en psycholinguistique (**[Miller et al. 87] [Miller 75] [Miller86] [Garett82] [FillenBaum et al. 65]**). Une conséquence directe de ces hypothèses sur l'implémentation de WordNet est le fait que les concepteurs ont séparé tous les mots selon leur catégorie syntaxique (verbes, adjectifs, noms, ...), au prix d'une certaine redondance. WordNet a été développé par le laboratoire de sciences cognitives à l'université de Princeton, sous la direction du Professeur Georges A. Miller. Le réseau contient approximativement 95600 formes de mots organisées en 70100 ensembles de synonymes, et est continuellement mis à jour.

Pour coder les notions sémantiques, les concepteurs ont considéré que les synonymes d'un mot peuvent aider à déterminer sa sémantique, et différencier le sens du mot. Ainsi chaque mot est toujours donné sous la forme d'un ensemble de synonymes. Par exemple, on aura $\{F_1,F_2,...,F_n\}$ comme enregistrement où $F_1,..,F_n$ sont les formes de mots correspondant à une signification donnée. Si un terme n'a pas de synonyme disponible, alors il pourra être accompagné d'une définition. En supposant que *board* n'a pas de synonyme, on pourra avoir

{board,(a person's meal, provided regularly for money)}

Cette courte définition va donc permettre de différencier le sens du mot *board* par rapport aux autres significations possibles.

Dans ce chapitre, nous décrivons les relations modélisées dans WordNet. Ensuite, nous expliquons la manière dont est implémenté ce

---

[10] *Georges A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller,* Introduction to WordNet : An On-Line Lexical Database, *http://www.cogsci.princeton.edu/~wn/*
WordNet et sa documentation complète peuvent être téléchargés depuis la même adresse.

réseau et la façon dont on l'utilise dans notre programme, pour terminer par quelques idées sur la façon de l'améliorer.

B.2. Les relations modélisées

Un grand nombre de relations sont modélisées au sein de WordNet. Certaines d'entre elles peuvent être valables pour certaines catégories de mots et pas pour d'autres. C'est le cas de la relation « est une sorte de » qui est valable uniquement pour les noms, et dans une certaine mesure pour les verbes, mais certainement pas pour les adjectifs et adverbes. Mais décrivons la majorité de ces relations :

Synonyme	Mots de même signification	{*maison,domicile*}
Antonyme	Mots de signification opposée	*monter/descendre*
Hyponyme	est une sorte de	*arbre* est hyponyme de *plante*
Hypernyme	à pour cas particulier	*plante* est un hypernyme de *arbre*
Métonyme	est une partie de	*aluminium* est un métonyme de *avion*
Holonyme	à comme composant	*arbre* est holonyme de *branche*
Attribut	est attribut de	*bec* est attribut de *canari*
Troponyme	est une manière de	*ramper* est troponyme de *marcher*
Implication	implique que	*ronfler* implique de *dormir*
Cause	est la cause de	*enseigner* est la cause d'*apprendre*

B.3. Remarques concernant les noms

Ces relations sont en général implémentées par un système de pointeurs entre ensemble de synonymes. En plus de ces pointeurs, les catégories syntaxiques des noms et des verbes sont classées selon une méthode permettant de trouver une certaine sémantique.

Les concepteurs ont choisi de construire une hiérarchie multiple, permettant de repérer avec plus de précision à quels concepts un mot se rapporte. Ainsi, lorsqu'on remonte les différentes hiérarchies, on aboutit à un des 25 « concepts généraux » indiquant clairement le contexte dans lequel on se trouve.

Voici ces 25 concepts :

{act, action, activity}
{animal, fauna}
{artifact}
{attribute, property}
{body,corpus}
{cognition,knowledge}
{communication}
{event, happening}
{feeling, emotion}
{food}
{group, collection}
{location, place}
{motive}
{natural object}
{natural phenomenon}
{person, human being}
{plant, flora}
{possession}
{process}
{quantity, amount}
{relation}
{shape}
{state, condition}
{substance}
{time}

Le nombre de noms présents dans chaque catégorie est très variable et un même nom peut se trouver dans différentes catégories.

Certaines limitations ont été introduites lors de l'implémentation. Par exemple, la relation « *est attribut de* » n'indique que les attributs étant eux-mêmes des noms. Ainsi, le mot *canari* possède l'attribut *bec* mais pas *jaune*. Cette limitation sera certainement levée lors d'une prochaine version de WordNet.

La conception de WordNet permet de différencier plusieurs types de relations de métonymie. Ainsi, on distingue trois types de relations :

- *est composant de*, ex. branche/arbre
- *est membre de*, ex. arbre/forêt
- *est la matière de*, ex. aluminum/avion

B.4. Remarques concernant les verbes

Comme pour la catégorie des noms, l'ensemble des verbes a été divisé en 15 catégories, largement sur base de critères sémantiques. Ainsi, les verbes sont également organisés en un certain nombre de classes de mots reprises ci-dessous :

1. Les verbes de fonctions du corps qui sont les verbes de fonctions habituellement réalisées par le corps (sweat, shiver, ...).
2. Les verbes de changement qui sont les verbes impliquant un changement d'état (become, alter...).
3. Les verbes de communication regroupent tous les verbes de communication verbale et non verbale (petition, order...).
4. Les verbes de compétition couvrent les domaines sémantiques du sport, les jeux et des luttes (face-off, duel...).

5. Les verbe de consommation incluent les verbes relatifs à l'ingestion, l'utilisation, l'exploitation, l'épandage et le partage (drink, eat...).
6. Les verbes de contact qui constituent le plus grande classe des verbes. La majeure partie de ces verbes est constituée des troponymes de quelques verbes de base (rub, grasp...).
7. Les verbes de cognition comprennent les verbes dénotant des actions cognitives (reasoning, judging...).
8. Les verbes de création sont encore divisés en trois catégories : création par un acte mental (invent...), création par des moyens artistiques (draw...), et enfin création à partir de matériaux divers (weave...).
9. Les verbes de mouvement dérivent tous de deux racines {move, make a movement} et {move, travel} (swim, fly...).
10. Les verbes d'émotion reprennent des verbes exprimant les émotions humaines (love, fear...).
11. Les verbes statiques qui sont des verbes pouvant indiquer une position statique (surround...)
12. Les verbes de perception comprennent tous les verbes référant à la perception par les cinq sens (spy, sniff...).
13. Les verbes de possession sont pour la plupart dérivés de trois concepts : {have, told, own}, {give, transfer}, et {take, receive}. Ces verbes dénotent le changement de possession ainsi que les états précédents ou suivants ce changement de possession (inherit, grant...).
14. Les verbes d'interaction sociale reprennent les verbes ayant rapport avec les différentes facettes de la vie sociale tel que la loi, la politique...(veto, excommunicate...).
15. Les verbes de temps comprennent les verbes ayant rapport a la météo (rain, thunder...).

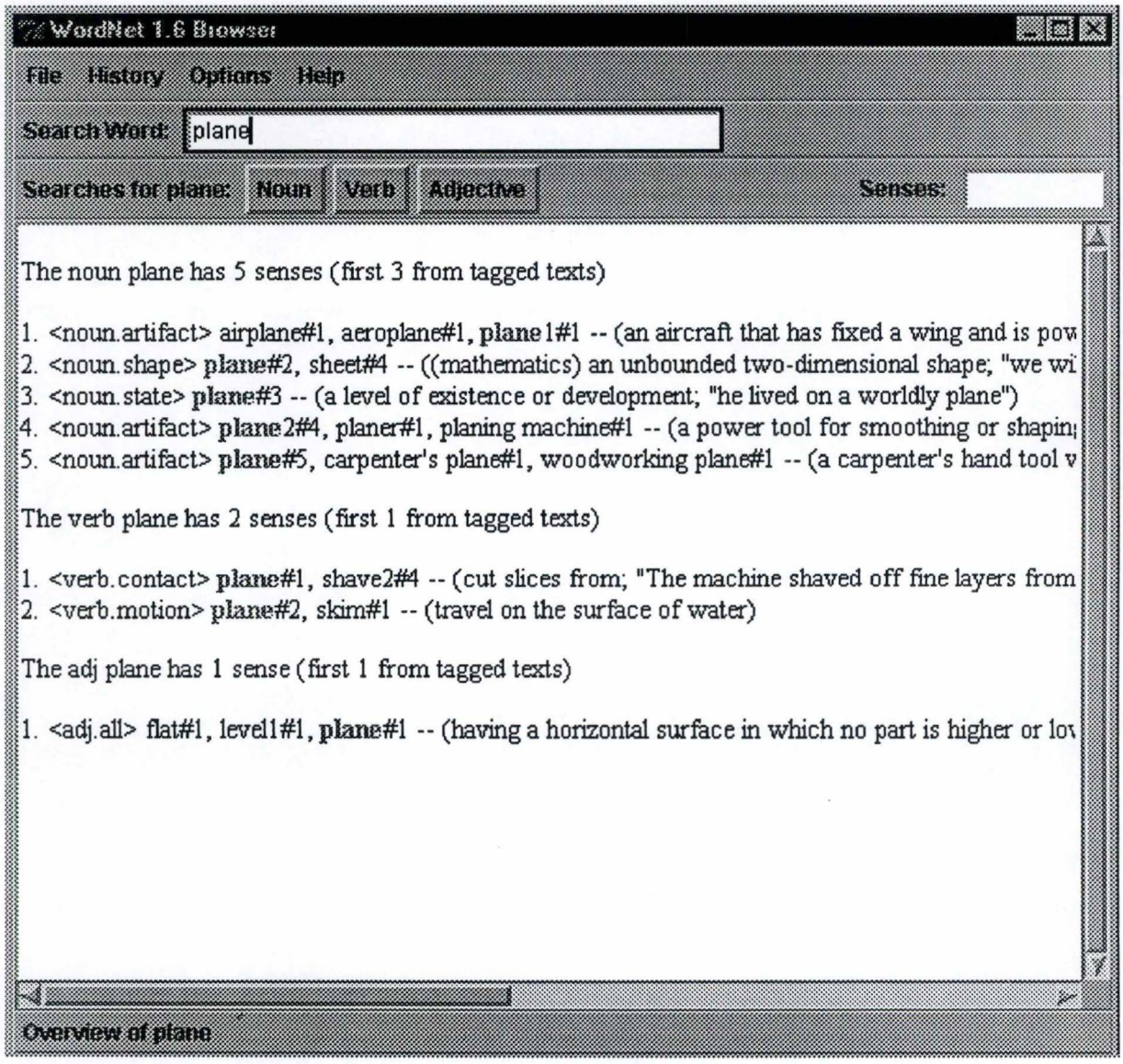
B.5. Implémentation de WordNet

On peut se procurer la base de données sous trois formes différentes : un package complet comprenant toutes les bases de données ainsi qu'une interface et un moteur d'accès permettant d'interroger ces bases de données, l'ensemble des fichiers sous forme ASCII, ou encore sous forme de clauses PROLOG. Pour illustrer les différentes relations que nous venons d'énoncer, et montrer la grande quantité d'information sémantique qu'il est possible d'extraire en utilisant les bases de données WordNet, nous avons commencé par indiquer les résultats fournis par le package complet. Suivant cela, nous

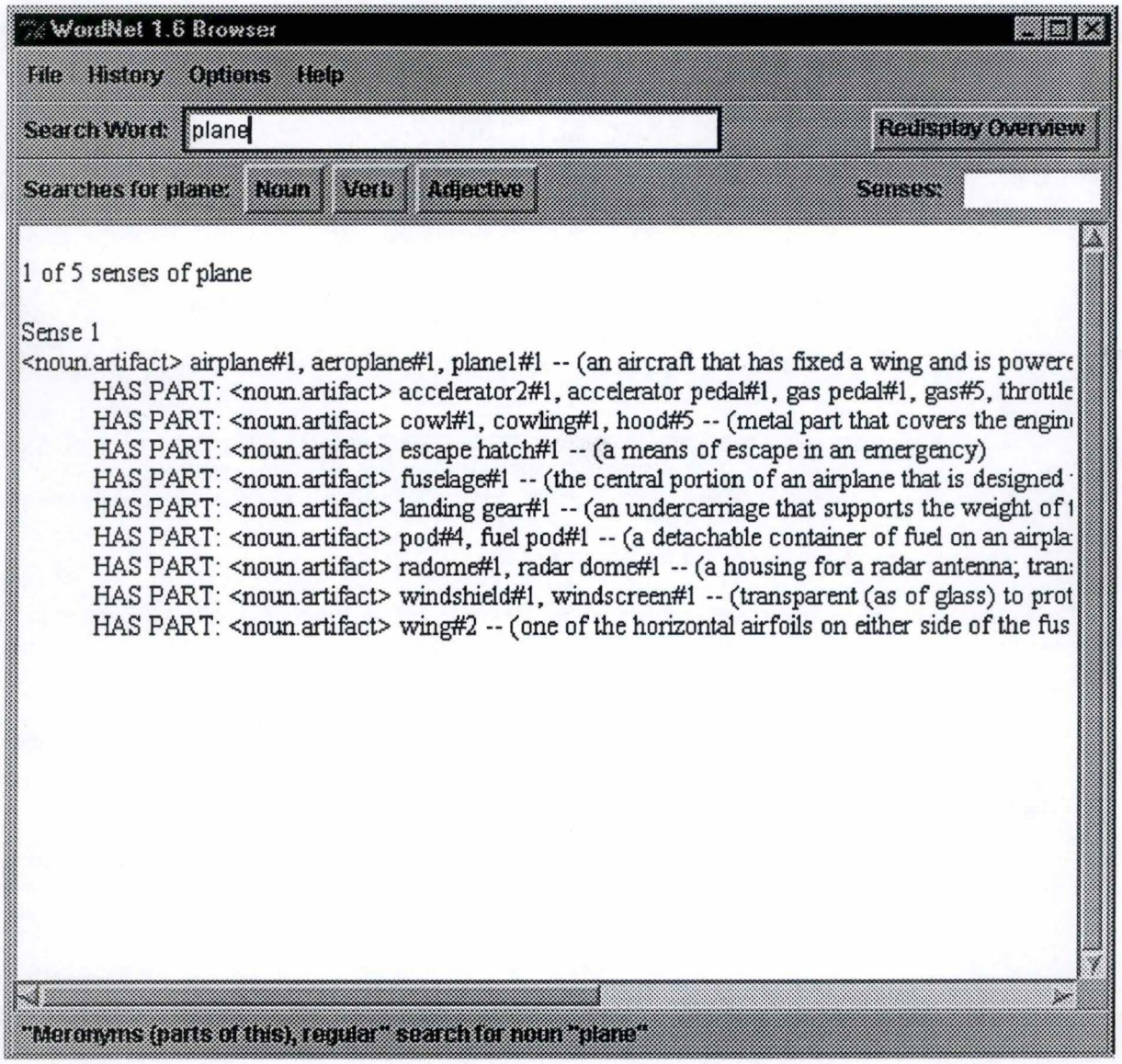
décrivons la structure de la base de données fournie sous forme de clauses PROLOG, dont nous avons utilisé un petit nombre de possibilités pour réaliser le programme durant notre stage.

a) Interface WordNet 1.6

Nous avons choisi de rechercher quelques informations sur le mot « plane ». Le premier écran ci-dessous nous indique les définitions fournies par la base de données :



Ensuite, nous pouvons choisir entre un grand nombre de relations. Nous voulons obtenir les composants de « plane » :



Cette base de données peut être comparée à un réseau sémantique, qui serait construit d'avance, ou encore à des « frames », couramment utilisés en intelligence artificielle. On comprend l'utilité qu'un tel système peut avoir dans la pratique. Les bases de données disponibles pourront permettre de gagner un temps précieux pour la mise au point de programmes nécessitant une période d'apprentissage. De même, de nombreux travaux ont été réalisés en ce qui concerne l'aide que WordNet peut fournir pour réduire l'ambiguïté lors de tâches d'analyse syntaxique **[Resnik]**.

b) Les bases de données PROLOG[11]

Voyons maintenant la manière dont les relations sont prises en compte par des clauses PROLOG.

Chaque fichier PROLOG contient l'information relative à une relation, ce qui rend l'utilisation assez claire. Le format général d'une ligne d'un fichier PROLOG est *operator(field1,...,field n)*. Listons les fichiers les plus importants :

Relations :

- **s**(*synset_id,w_num,word,ss_type,sense_number,tag_state)* représente un ensemble de synonymes. Il existe une ligne par sens d'un mot.
- **hyp**(*synset_id,synset_id)* spécifie que le second ensemble de synonymes est un hypernyme du premier.
- **ent**(*synset_id,synset_id)* spécifie que le second ensemble de synonymes est une implication du premier.
- **sim**(*synset_id,synset_id)* spécifie que le second ensemble de synonymes est similaire en signification au premier.
- **mm**(*synset_id,synset_id*) spécifie que le second ensemble de synonymes est un métonyme (membre) du premier.
- **ms**(*synset_id,synset_id*) spécifie que le second ensemble de synonymes est un métonyme (substance) du premier.
- **mp**(*synset_id,synset_id*) spécifie que le second ensemble de synonymes est un métonyme (partie) du premier.
- **cs**(*synset_id,synset_id*) spécifie que le second ensemble de synonymes est une cause du premier. Cette relation est seulement valable pour les verbes.
- **at**(*synset_id,synset_id*) spécifie une relation entre noms et adjectifs dans laquelle l'adjectif est une valeur du nom.
- **ant**(*synset_id,w_num,synset_id,w_num*) spécifie que les mots désignés sont des antonymes. *W_num* permet de différencier les mots d'un ensemble de synonymes.

---

[11] *On peut télécharger à l'adresse indiquée plus haut un ensemble de fichiers nommés* ***wn_*.pl*** *contenant la base de données WordNet dans un format lisible par PROLOG.*

Signification des paramètres :

- *synset_id* est un champ de neuf chiffres. Le premier chiffre décrit la catégorie syntaxique de l'ensemble de synonymes et les huit chiffres restants définissent le déplacement à effectuer dans le fichier pour trouver le début de l'ensemble. La catégorie syntaxique est codée comme suit :
  - 1 = nom
  - 2 = verbe
  - 3 = adjectif
  - 4 = adverbe
- *ss_type* est un code d'un caractère utilisé pour définir le type de *synset* et est défini comme suit :
  - n = nom
  - v = verbe
  - a = adjectif
  - s = adjectif satellite
  - r = adverbe
- *sense_number* spécifie le numéro du sens du mot, dans la catégorie syntaxique encodée dans le *synset_id*.
- *word* est le mot codé, où les espaces sont remplacés par des '_'.

B.6. Utilisation de WordNet dans notre programme

Nous avons simplement utilisé les relations d'hypernymie et de synonymie. On pourrait certainement utiliser WordNet d'une manière beaucoup plus intense, en combinaison avec d'autres techniques (statistiques...) pour arriver à de meilleurs résultats. Les questions relatives à l'amélioration du programme initial se trouvent dans le dernier chapitre.

On notera cependant que nous n'avons pas utilisé les bases de données téléchargées telles quelles. En effet, les fichiers de faits PROLOG contiennent l'équivalent de milliers de pages de texte. Il n'était donc pas possible de les placer tous en mémoire vive. De même, on ne pouvait pas lire à chaque fois ces fichiers pour trouver de l'information, sous peine d'obtenir des performances désastreuses. Nous avons alors cherché dans la documentation s'il existait des librairies permettant de créer des bases de données indexées, ce qui était le cas. Nous avons indexé les champs *synset_id*, pour pouvoir nous déplacer plus rapidement parmi les *synsets*, ce qui nous permettait d'utiliser WordNet d'une manière transparente lors de conversations avec des utilisateurs.

Chapitre IV
Perspectives

A. Introduction

Nous avons exposé jusqu'à présent quelques techniques permettant de modéliser certains aspects du langage naturel. Ensuite, nous avons exposé l'architecture du programme réalisé, en spécifiant le fait que nous avions juste réalisé un « moteur principal » et que la prochaine étape serait l'intégration de modules relatifs à l'apprentissage du langage.

Ce chapitre reprend plusieurs suggestions quant à des améliorations souhaitables à apporter à notre programme. Certaines de ces idées viennent directement des techniques d'apprentissage présentées au chapitre 3, d'autres nous ont été conseillées par les chercheurs du département de Flinders University. Enfin, certaines idées sont personnelles. Il est clair que ces méthodes ne doivent pas être toutes utilisées à la fois, mais peuvent être testées, comparées et combinées pour obtenir les meilleurs résultats possibles.

B. Correction orthographique

B.1. Enregistrement des erreurs les plus fréquentes

Il serait intéressant d'établir des statistiques lorsque le module de correction orthographique est sollicité. En effet, la version actuelle n'enregistre pas les mots inconnus pour lesquels une correction a été effectuée. Ainsi, si deux minutes plus tard, la correction orthographique est rappelée pour le même mot, un temps précieux sera perdu pour cette nouvelle recherche.

Il serait alors utile d'établir une base de données des mots corrigés. Celle-ci comporterait par exemple un enregistrement comprenant le mot fautif, ses statistiques d'apparition, et le mot corrigé.

Pour éviter de saturer l'espace disque disponible, il faudra penser à éliminer certains mots, par exemple sur base de leur fréquence d'apparition.

B.2. Détection des « ensembles de confusion »

Un certain nombre de mots anglais ont une orthographe très semblable et sont couramment intervertis. Voici quelques exemples de confusion : {their, there, they're}, {weather, whether}, {country, contry}...

Une première méthode pour supprimer l'ambiguïté lorsqu'on rencontre un mot présent dans un ensemble de confusion serait de consulter les trigrammes de mots, et déterminer si notre cas (le mot ambigu et son contexte) apparaît dans ces trigrammes. Ce système nécessite d'avoir lu préalablement un grand nombre de corpora, pour pouvoir prendre en compte le plus de cas possible. Les ressources utilisées (place de la base de données sur les disques) risquent d'être énormes.

Une seconde méthode serait de remplacer le mot ambigu par chacun des autres mots de l'ensemble de confusion dans lequel il se trouve. Pour chaque nouvelle phrase ainsi formée, il est possible de calculer sa probabilité d'apparition si on possède un modèle Markovien du langage. On remplacerait alors le mot ambigu par celui qui figure dans la phrase la plus probable.

C. *Génération des commentaires*

C'est la génération de commentaires qui nous permet de construire le « fil conducteur » de la conversation et de changer de sujet si les commentaires pré-définis viennent à manquer. Actuellement, ce fil conducteur est assez rudimentaire puisqu'après avoir déterminé les mots importants, nous choisissons un des nombreux commentaires pré-définis à notre disposition.

Une meilleure solution serait de modéliser le comportement du langage et établir un modèle de Markov du langage tel que nous l'avons défini dans le chapitre 2. Comme nous l'avons déjà indiqué, la détermination des états peut se faire à l'aide de la technique de classification.

Après chaque introduction d'un commentaire d'un juge, nous pouvons générer des répliques à l'aide du modèle de Markov du langage. Pour tenter de rester dans le contexte de la conversation, nous imposons certains mots-clés dans le commentaire généré. **[Hutchens et al. 98]** a utilisé un système assez semblable pour son programme MegaHal, présenté au Loebner Prize à Sydney.

La chaîne de Markov peut générer un très grand nombre de répliques. Comment déterminer celle qui est la plus appropriée dans la conversation ? Pour cela, **[Hutchens et al. 98]** utilise l'entropie croisée (voir chapitre 2.A.) et choisit le commentaire qui est le plus « étonnant » (qui possède la plus haute entropie) par rapport au modèle de Markov présent. Nous pensons faire exactement le contraire et choisir le commentaire le plus réaliste par rapport au modèle que nous possédons, c'est à dire qui possède l'entropie la plus basse. Remarquons que la formule de l'entropie croisée décrite dans le chapitre 2 calcule l'entropie d'un modèle, c'est à dire la ressemblance de ce modèle avec la réalité. Dans le cas présent, nous devons utiliser l'entropie croisée d'un message, c'est à dire la vraisemblance de ce message avec un message généré par le « modèle réel » du langage.

D. *Utilisation de sources d'information variées*

Comme nous l'avons déjà indiqué, la version actuelle de notre programme ne génère que des commentaires définis à l'avance, encodés lors du développement du programme. Il serait certainement intéressant de varier les sources d'information qui permettent de générer des commentaires. Ainsi, après avoir construit un modèle de Markov du langage, l'entraînement de celui-ci pourrait se faire selon trois sources d'information distinctes :

- Des bribes de conversation enregistrées de façon manuelle.
- La lecture de corpus, permettant d'affiner sans cesse notre modèle.
- Des connaissances générales par lecture d'informations techniques (encyclopédies, Internet...).

Il serait également très intéressant de pouvoir tenir compte de conversations tenues entre le programme et des personnes d'horizon variés. Une façon de permettre cela serait de laisser le programme à disposition pour que des gens l'utilisent. Le meilleur endroit de mise à disposition du programme serait bien sûr de le rendre disponible sur Internet et enregistrer toutes les conversations effectuées.

E. *Recherche du sujet de la phrase*

Comme nous l'avons vu dans le chapitre consacré à la classification (voir chapitre 2.B.), il est possible de générer une grammaire. Grâce à l'utilisation d'une grammaire, nous pouvons récupérer la structure syntaxique (voir chapitre 1.D.) du message et ainsi mettre en évidence des informations telles que le sujet, le complément d'objet, ... et ce grâce aux régles de dépendance (**[Deville89]**).

La méthode que nous utilisons actuellement dans notre application pose certains problèmes. Les mots-clés que nous avons sont traités dans l'ordre de leur apparition. Si nous avons la phrase : « In my village, my friends are very curious.», nous aimerions formuler une réponse sur le sujet *friends*. Le mot-clé *village* est sujet (thème pré-défini) et est placé avant le mot-clé *friends* dans la phrase. Donc, notre application formule une réponse basée sur le sujet *village*.

Par contre, les éléments syntaxiques peuvent être utiles pour remédier à ce problème. En effet, si nous connaissons le sujet et le complément d'objet du message, nous pouvons éviter de prendre en compte des substantifs inutiles (comme le mot-clé *village*) et ainsi formuler une réponse plus précise.

F. Recherche de la catégorie syntaxique d'un mot.

La tâche importante de notre « moteur principal » est la recherche de mots-clés sujets et de mots-clés non sujets. Comme mots-clés, nous recherchons particulièrement les substantifs. A chaque analyse d'un mot du message encodé par le juge, nous comparons celui-ci à une liste de faits PROLOG ; s'il n'est pas repris dans celle-ci nous le considérons comme mot-clé. Dans cette liste de faits, nous avons repris des pronoms. Cependant, il serait extrêmement long de devoir créer un fait pour chaque mot de la langue anglaise qui n'est pas un substantif.

Deux possibilités se présentent à nous pour éviter ce grand nombre de faits PROLOG. Une première solution est de se référer aux résultats obtenus lors d'un processus de classification (voir chapitre 2.B.). Ce processus agit sur un ou plusieurs corpus et construit les classes syntaxiques. Cependant, un mot peut appartenir à plusieurs catégories syntaxiques. Or, ce phénomène d'ambiguïté syntaxique n'est pas solutionné dans cette approche.

L'autre solution est d'utiliser un réseau de neurones (voir chapitre 2). Nous commençons par « tagger[1] » les mots d'un corpus puis nous répertorions l'ensemble des bigrammes de ce corpus qui nous servirons pour l'entraînement. Lors de l'utilisation du réseau, si on lui donne un masque d'entrée[2] (sous forme de catégorie syntaxique), il nous rend un masque de sortie qui correspond à la catégorie syntaxique du mot suivant. Ainsi, lors de la recherche de mots-clés, nous saurons directement si le mot suivant à de grandes chances d'être un substantif ou non.

---

[1] *Associer un mot avec sa catégorie syntaxique.*

[2] *Le principe de codage d'une catégorie syntaxique comme masque d'entrée d'un réseau est expliqué au chapitre 2.C.*

G. Suppression de l'ambiguïté des mots.

Dans cette partie, nous allons décrire deux types d'ambiguïté. Il peut exister de l'ambiguïté par rapport aux sens des mots. Un mot peut posséder plusieurs sens et il nous faut choisir le sens qui est utilisé dans la phrase. La seconde ambiguïté est la relative aux catégories syntaxiques. Un mot peut appartenir à plusieurs catégories syntaxiques et il nous faut sélectionner celle à laquelle le mot appartient dans la phrase.

En ce qui concerne l'ambiguïté entre les sens d'un mot, nous avons remarqué que cela peut provoquer l'affichage par le programme d'un commentaire qui n'est pas du tout adéquat dans le contexte de la conversation en cours. Le sujet trouvé par le programme peut se révéler avoir plusieurs significations. Il y a problème car le commentaire que nous avons pré-défini ne concerne pas nécessairement le sens qui est utilisé dans la conversation actuelle. Donc, la réponse que nous fournissons peut être tout à fait inadéquate.

Une possibilité pour supprimer cet inconvénient est citée dans **[Li et al.]**. Ils proposent de lever l'ambiguïté relative au sens d'un mot en utilisant le verbe associé à ce mot dans la phrase. Pour trouver le verbe le plus proche d'un mot, ils utilisent les méthodes de parsing (voir chapitre 1.D.). Nous allons rechercher, dans un corpus, un autre mot associé à ce verbe. Ensuite, nous calculons cette distance de similarité entre ce mot et le mot ambigu. Pour calculer une distance de similarité, nous allons naviguer parmi les synonymes, hypernymes, ... qui peuvent permettre de retrouver un des sens du mot ambigu. Chaque relation sémantique de WordNet est pondérée (voir **[Li et al.]**). Lorsque nous avons trouvé la plus petite distance entre le mot et un des sens du mot ambigu, alors nous pouvons dire que c'est ce sens qui est employé dans la phrase. Si aucun mot associé au verbe n'a été trouvé, d'autres possibilités sont envisagées dans **[Li et al.]**.

Ainsi, nous pouvons savoir si le commentaire pré-défini s'accorde avec ce sens ou non. Nous pouvons également améliorer notre application en prévoyant plusieurs commentaires pour les différents sens possibles d'un mot.

En ce qui concerne l'ambiguïté relative aux catégories syntaxiques, cela nous pose problème lors de la recherche des substantifs comme mots-

clés. Il se peut que l'on trouve un mot appartenant à plusieurs catégories syntaxiques. Une solution pour trouver la catégorie correcte est décrite par l'utilisation des réseaux de neurones ou des chaînes de Markov (voir chapitre 2). Nous avons un corpus « taggé » à notre disposition et nous entraînons le réseau avec des « sixgrammes[3] ». Lors de l'exécution du réseau, on lui donne un masque d'entrée (quatre mots précédents, le mot ambigu et le mot suivant) et il nous rend un masque de sortie qui correspond à la catégorie syntaxique du mot ambigu pour cette phrase. Ainsi, lors de la recherche de mots-clés, nous saurons directement si ce mot est un substantif ou non.

H. Apprentissage de commentaires.

Chaque commentaire que nous formulons comme réponse à un message du juge est conservé dans un fichier. Nous pensons que c'est une perte de temps que d'ajouter des commentaires manuellement. De plus, si un juge aborde plusieurs fois le même sujet, il est fort probable que celui-ci ait plusieurs fois la même réponse à cause de la technique du *carrousel* (voir chapitre 3).

C'est pourquoi une autre amélioration possible est d'augmenter de manière automatique le nombre des commentaires pré-définis. Pour réaliser cette fonctionnalité, nous pensons qu'il serait intéressant de conserver certaines des affirmations encodées par les utilisateurs. Chaque message encodé serait analysé et les mots-clés sujets et les mots-clés seraient extraits.

Il est alors possible d'ajouter un nouveau fait PROLOG pour ce commentaire. De cette manière, l'ensemble de nos commentaires sera de plus en plus important.

I. Prise en compte du commentaire précédent

En examinant l'architecture générale de l'application, nous avons pu constater que l'absence de mots-clés sujets ne nous permet pas toujours de fournir un commentaire adapté au fil de la conversation. Lorsque ce cas se présente, nous produisons un commentaire de requête de complément

---

[3] *Le « sixgramme » est formé des quatre mots précédents, du mot ambigu et du mot suivant*

d'informations tel que « Why do you think that? » ou « Can you talk more abou that ... ».

Il peut être intéressant d'ajouter une autre possibilité de génération d'un commentaire lorsque nous n'avons aucun mot-sujet à notre disposition. Nous pouvons utiliser une autre technique qui consiste à garder l'historique des sujets exposés depuis le début de la conversation. Ainsi, lorsque le cas se présente, nous pouvons ignorer le message et orienter la conversation sur un autre sujet abordé précédemment ou sur un autre sujet de notre choix.

J. Réponse à des calculs mathématiques ou à des questions relatives à l'heure et à la date.

Lorsque nous conversons avec d'autres personnes, il nous arrive de résoudre de simples calculs mathématiques. Une telle fonctionnalité à notre programme permettrait de mieux simuler le comportement humain. Il faut, pour cela extraire les nombres et identifier les opérations sur ceux-ci. Ensuite nous formulons la réponse à ce calcul.

De même, pour l'heure et la date, nous devons analyser le message encodé par le juge en extrayant les mots significatifs (jour de la semaine, période de la journée, ...) qui permettent de cerner la nature de la question. Nous devons également avoir à notre disposition toutes les informations nécessaires sur la date et l'heure du moment.

Conclusion

Ce mémoire nous a permis de nous familiariser avec un domaine relativement inconnu dans notre formation. C'est la raison pour laquelle nous avons choisi de présenter les techniques qui nous semblaient importantes dans le traitement du langage naturel.

Il était peut être difficile pour les lecteurs de se rendre compte de l'utilité des méthodes décrites ainsi que de la liaison entre les différents chapitres. Le chapitre I nous a permis de fixer les bases théoriques utiles pour la compréhension du reste du mémoire.

Dans le chapitre II, nous avons exposé les techniques d'apprentissage du langage naturel par les machines. Il semble que la technique de l'apprentissage statistique du langage soit fortement utilisée dans la pratique. Pourtant, nous pensons qu'il serait intéressant d'utiliser plusieurs de ces techniques d'une manière complémentaire pour obtenir des résultats optimaux.

Le chapitre III présentait l'application participant à la compétition (*Loebner Prize*). Seules les méthodes implémentées étaient détaillées dans ce chapitre. Remarquons également que notre application était la seule à disposer de *WordNet*. Cette utilisation s'est révélée efficace car cet outil nous a offert des moyens pour sans cesse raffiner notre commentaire. De nombreuse autres applications pourraient être réalisées avec WordNet.

Nous avons expliqué alors dans le chapitre IV différentes possibilités d'amélioration. Malheureusement, nous n'avons pu les implémenter nous-mêmes, mais d'autres étudiants continuent notre travail en ce moment même.

Notre stage nous a apporté de nombreux bénéfices, autres que ceux découlant du seul fait de travailler dans un nouveau domaine. En effet, l'utilisation de PROLOG a perfectionné notre connaissance de ce langage et nous a montré les avantages liés à son utilisation dans certains domaines, notamment le traitement des chaînes de caractères.

Notre participation au Loebner Prize fût une expérience enrichissante. Elle nous a donné l'occasion de travailler sous forte pression, peut être plus proche de celle que nous rencontrerons durant notre vie professionnelle. Elle nous a également permis de nous rendre compte des résultats des différents participants et de constater qu'il reste bien des progrès à réaliser avant de pouvoir déclarer une machine intelligente.

Bien que nos résultats lors de cette compétition soient modestes, ceux-ci nous semblent raisonnables compte tenu du temps dont nous disposions pour la réalisation de l'application. Nous pouvons aussi mesurer notre relative faiblesse par rapport aux autres participants qui, pour la majorité, travaillent dans le traitement du langage naturel depuis de nombreuses années.

Enfin, nous espérons que notre contribution dans la réalisation du programme aidera les prochains participants du Loebner Prize à implémenter les améliorations nécessaires en vue d'un meilleur classement.

Bibliographie

[Aleksander et al. 90]
Igor Aleksander, Helen Morton, *An intoduction to neural computing*, Chapman and Hall, 1990,pp. 92-110.

[Allsch93]
J.-M. ALLIOT, T. SCHIEX, *Intelligence artificielle & informatique théorique*, Cépaduès-éditions, 111, rue Nicolas-Vauquelin - 31100 Toulouse, 1993,

[Anderson83]
J.R. Anderson, *The architecture of cognition*, Cambridge, Mass. : Harvard University Press, 1983

[Baum72]
L.E. Baum, « *An inequality and associated maximization technique in statistical estimation for probabilistic function of Markov process* », Inequalities 3,1972, 1-8

[Benello et al. 89]
J. Benello, A. W. Mackie, J. A. Anderson, *Syntactic category disambiguation with neural networks*, Computer Speech and Language, vol. 3, 1989, pp. 203-217.

[Berleur91]
Jacques BERLEUR s.j., « Langage Naturel et Intelligence Artificielle : Quelques Réflexions Epistémologiques »

[Bloom93]
Paul Bloom, *Language acquisition : core readings*, Harvester Wheatsheaf New York, 1993

[Bow92]
Sing-Tze BOW, *Pattern Recognition and Image Preprocessing*, Marcel Dekker, 1992, pp. 88-142.

[Brill]
Eric Brill, « Automatic Grammar Induction and Parsing Free Text : A Transformation-Based Approach », Department of Computer and Information Science, University of Pennsylvania, brill@unagi.cis.upenn.edu

[Brill92]
Eric Brill, « A simple rule-based part of speech tagger », in *Proceedings of the ThirdConference on Applied Natural Language Processing*, ACL, Trento, Italy, 1992

[Brill93]
Eric Brill, « A Corpus-Based Approach to Language Learning », PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1993.

[Brown et al. 63]
Brown, Roger Brown, and Colin Fraser, « The acquisition of syntax », in C.N. Cofer and Barbara S. Musgrave, *Verbal Behavior and Learning : Problems and Processes*, Mc Graw-Hill, 1963, pp.160

[Caramazza et al. 78]
A. Caramazza et R.S. Berndt, « Semantic and Syntactic Processes in Aphasia : A Review of the Literature », *Psychological Bulletin* 85, 1978, pp.898-918

[Charniak93]
Eugène CHARNIAK, *Statistical Language Learning*, The MIT Press, Cambridge, Massachusetts, 1993, pp. 27-52.

[Charniak93 b]
Eugène CHARNIAK, *Statistical Language Learning*, The MIT Press, Cambridge, Massachusetts, 1993, pp. 65-68

[Charniak93c]
Eugène CHARNIAK, *Statistical Language Learning*, The MIT Press, Cambridge, Massachusetts, pp. 1-20, 1993

[Charniak93d]
Eugène CHARNIAK, *Statistical Language Learning*, The MIT Press, Cambridge, Massachusetts, pp. 135-145, 1993

[Collins et al. 69]
A.M. Collins, M.R. Quillian, « Retrieval Time From Semantic Memory », *Journal of Verbal Behavior and Verbal Learning 8*, 1969 , pp.240-247

[De Saussure71]
Ferdinand de Saussure, *Cours de linguistique générale*, Paris, Payot, 3ème édition, 1971

[Deville89]
Guy Deville, *Modelization of task-oriented utterances in a man-machine dialogue system*, Ph.D Thesis UIA, Antwerp, 1989.

[Deville98]
Guy Deville, *Introduction aux méthodes et concepts essentiels d'ingénierie linguistique*, Cours de 2ème licence et 3ème maîtrise, 1998.

[Ducrot et al. 72]
Oswald Ducrot, Tzvetan Todorov, *Dictionnaire encyclopédique des sciences du langage*, Editions du Seuil, Paris, 1972, pp. 49-64.

[Eisenson et al. 63]
Vetter, H.J. and R.W. Howell, « Theories of Language Acquisition », *Jnl of Psycholinguistic Res.*, Vol.1, N° 1, 1971, pp. 34ff.

[Ency9]
La grande encyclopédie du 20ème siècle, Editions Christophe Colomb, Volume 9, 1988, pp. 1932-1933.

[Ency11]
La grande encyclopédie du 20ème siècle, Editions Christophe Colomb, Volume 11, 1988, pp. 2508-2511.

[Golding et al. 96]
Andrew R. Golding and Yves Schabes, « Combining Trigram-based and Features-based Methods for Context-Sensitive Spelling Correction, Mitsubishi Electric Research Laboratories, 201 Brodway, Cambridge, MA 02139, disponible à http ://www.merl.com/reports/index.html.

[Hanson87]
S. J. Hanson, J. Kegl, *PARSNIP : A Connectionist network that learns natural language grammar from exposure to natural language sentences*, in Proc. of the Ninth Annual Conference of the Cognitive Science Society, Lawrence Erlbaum Associates, Hillsdale, 1987, pp. 106-119.

[Hutchens et al. 98]

Jason L. Hutchens, Michael D. Alder, « Introducing MegaHAL », *New Methods in Language Processing and Computational Language Learning (NeMLaP3/CoNLL98),* Editor : David M.W. Powers, 1998

[Korkmaz97]

Emin Erkan Korkmaz, *A Method for Improving Automatic Word Categorization*, Ph.D Dissertation, The Middle East Technical University, 1997, pp. 11-30.

[Lemoigne86]

Jean-Louis Le Moigne, *Intelligence des Mécanismes, Mécanismes de l'Intelligence : Intelligence Artificielle et Sciences de la Cognition*, Fondation Diderot Fayard, France, 1986, pp. 15-55.

[Levinson et al. 83]

S.E. Levinson, L.R. Rabiner, and M.M. Sondhi, « An introduction to the application of the theory of probalistic functions of a Markov process to automatic speech recognition », *The Bell System Technical Journal* 62, 1983, pp. 1035-1074

[Li al.]

Xiaobin Li, Stan Szpakowicz, Stan Matwin, *A WordNet-based Algorithm for Word Sense Disambiguation.*

[McKoon et al.]

G. McKoon et R. Ratcliff, « priming in item recognition : The organization of propositions in memory for text. », *Journal of Verbal Learning and Verbal Behaviour* 19, pp. 369-386

[Malsburg73]

C. VON DER MALSBURG, « Self-Organization of Orientation Selective Cells in the Striate Cortex », *Kybermetik*, Vol. 14 (1973), pp. 85-100

[Miller et al. 93]

George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller, *Introduction to WordNet : An On-Line Lexical Database*, http ://www.wordnet.com, 5papers.ps, 1993

[Morgan et al. 91]

David P. Morgan, Christopher L. Scofield, *Neural Networks and Speech Processing*, Kluwer Academic Publishers, 1991, pp. 245-288.

[Napoli96]

Donna Jo NAPOLI, *Linguistics*, Oxford Univeristy Press, New York, 1996, pp. 3-21.

[Nakamura et al. 89]
M. Nakamura, K. Shikano, *A Study of English word category prediction based on neural networks*, in Proc. ICASSP, Glasgow, Scotland, May 1989, pp. 731-734.

[Nivelles et al. 90]
Philippe Nivelles, Vincent Thiry, *Les réseaux neuronaux : Approche théorique et Applications*, Mémoire en Informatique, 1990, pp. 61-82.

[Powers et al. 89a]
David M.W. POWERS, Christopher C.R. TURK, *Machine Learning of Natural Language*, Springer-Verlag, London , 1989, p. 1

[Powers et al. 89b]
David M.W. POWERS, Christopher C.R. TURK, *Machine Learning of Natural Language*, Springer-Verlag, London , 1989, p. 161 - 182

[Powers et al. 89c]
David M.W. POWERS, Christopher C.R. TURK, *Machine Learning of Natural Language*, Springer-Verlag, London , 1989, p. 32

[Powers et al. 89d]
David M.W. POWERS, Christopher C.R. TURK, *Machine Learning of Natural Language*, Springer-Verlag, London , 1989, p. 34-40

[Powers et al. 89e]
David M.W. POWERS, Christopher C.R. TURK, *Machine Learning of Natural Language*, Springer-Verlag, London , 1989, p. 60

[Powers97]
David M. W. POWERS, « Unsupervised Learning of Linguistic Structure : An Empirical Evaluation», *International Journal of Corpus Linguistics*, Volume 2, Numéro 1, 1997, 91-131.

[Powers97 b]
David M.W. POWERS, « Learning and Application of Differential Grammars », *CoNLL97 : Computational Natural Language Learning,* ACL pp. 88-96

[Quillian67]
M. R. Quillian, « Word Concepts : A theory and Simulation of Some Basic Semantic Capabilities. », *Behavorial Science* 12, 1967, pp. 410-430

[Quillian68]
M.R. Quillian, « Semantic Memory » In Minsky, M. (ed.), *Semantic Information Processing*, Cambridge, Mass. : MIT Press, 1968

[Resnik]
Philip Resnik, *Disambiguating Noun Grouping with Respect to WordNet Senses*, Sun Microsystems Laboratories, Two Elizabeth Drive,

[Sachs67]
J.S. Sachs, « Recognition memory for synctatic and semantic aspects of connected discourse. », *Perception and Psychophysics* 2, pp. 437-442

[Samuelsson et al. 97}
Christer Samuelsson, Brigitte Krenn, *The Linguist's Guide to Statistics »*

[Schifferdecker94]
Schifferdecker, *Finding Structure in Language*, Diplom Thesis, University of Karlsruhe, 1994, pp. 41-68.

[Stillings et al. 87a]
Neil A. STILLINGS, Mark H. FEINSTEIN, Jay L. GARFIELD, Edwina L. RISSLAND, David A. ROSENBAUM, Steven E.WEISLER, Lynne BAKER-WARD, *Cognitive science, an introduction*, MIT Press, Massachusetts Institute of technology, 1987, p.20

[Stillings et al. 87b]
Neil A. STILLINGS, Mark H. FEINSTEIN, Jay L. GARFIELD, Edwina L. RISSLAND, David A. ROSENBAUM, Steven E.WEISLER, Lynne BAKER-WARD, *Cognitive science, an introduction*, MIT Press, Massachusetts Institute of technology, 1987, p.31-47

[Stillings et al. b]
Neil A. STILLINGS, Mark H. FEINSTEIN, Jay L. GARFIELD, Edwina L. RISSLAND, David A. ROSENBAUM, Steven E.WEISLER, Lynne BAKER-WARD, *Cognitive science, an introduction*, MIT Press, Massachusetts Institute of technology, 1987, p.56-59

[Sowa84]
J. F. SOWA, *Conceptual Structures : Information Processing in Mind and Machine*, Addison-Wesley, Etats-Unis, 1984, pp. 211-264.

[Vihman96]
Marylin May Vihman, *Phonological Development : The Origins of Language in the Child*, Blackwell Publishers, Massachusetts, 1996, pp. 3-11.

[Vogt]
Christopher C. Vogt, *Word Prediction Using a Neural Net.*

[Yule96a]

George Yule, *The study of language*, Cambridge University Press, Second edition, 1996, pp. 40-50.

[Yule96b]
George Yule, *The study of language*, Cambridge University Press, Second edition, 1996, pp. 53-57.

[Yule96c]
George Yule, *The study of language*, Cambridge University Press, Second edition, 1996, pp. 74-79.

[Yule96d]
George Yule, *The study of language*, Cambridge University Press, Second edition, 1996, pp. 86-110.

[Yule96e]
George Yule, *The study of language*, Cambridge University Press, Second edition, 1996, pp. 114-130.

[Weizenbaum81]
J. Weizenbaum, *Puissance de l'ordinateur et raisin de l'homme*, Editions de l'informatique, 1981.

Annexes

A. Code du programme

1. Programme principal

```
:- link_foreign_resource(sl,'sl.pl',dynamic,['sl.c'],[],[]).
foreign_resource(sl,[sleep_c]).
foreign(sleep_c,c,sleep_p(+integer)).
:- load_foreign_resource(sl).

:- sequential.
:- parallel paral/5.

%% ********************************************************************
%% find_syn_hyp(KeyWords,Sent,Lstay,SubList,Bd1,Bd2) takes the first 4 %%
keywords and search corresponding subjects,if found
%% ********************************************************************
find_syn_hyp([],Input,Lstay,[],Bd1,Bd2) :-not(predef(Input,_)),Lstay=[].
find_syn_hyp(NewKeys,Input,Lstay,[predef],Bd1,Bd2) :-
predef(Input,_),Lstay=[].
find_syn_hyp(NewKeys,Input,Lstay,Lsub,Bd1,Bd2) :-
not(NewKeys=[]),not(predef(Input,_)),subllist(NewKeys,Keys,0),find_syn(Keys,L
t,Bd1,Bd2),sort_sub(Lt,Lsub1,Lnosub1),find_hyp(Lnosub1,Rep,Bd1,Bd2),sort_sub(
Rep,Lsub2,Lnosub2),append(Lsub1,Lsub2,Lsub),del_list(Keys,NewKeys,Lstay).

paral(St,Lstay,Skeys,Lsub,Bd1,Bd2) :-
find_syn(Lstay,Lt,Bd1,Bd2),sort_sub(Lt,Lsub1,Lnosub1),find_hyp(Lnosub1,Rep,Bd
1,Bd2),sort_sub(Rep,Lsub2,Lnosub2),append(Lsub1,Lsub2,Lsub).
paral(St,Lstay,Skeys,Hkeys,Bd1,Bd2) :- gener_he(St,1).

%% ********************************************************************
%% group between subjects,no subjects and keywords for which a subject %% is
found
%% ********************************************************************
sort_sub([],[],[]).
```

```
sort_sub([[H,nosub]|T],X,[H|Y]) :- sort_sub(T,X,Y).
sort_sub([[H,Z]|T],[Z|X],Y) :- not(Z=nosub),sort_sub(T,X,Y).

%%*********************************************************
%% ouv_bd open databases necessary
%%*********************************************************
ouv_bd(Bd1,Bd2,Bd3,Bd4) :-
use_module(library(db)),db_open(db_eli_s,read,Bd1),db_open(db_eli_hyp,read,Bd
2),db_open(db_prox,read,Bd3),db_open(db_tril,read,Bd4).

%% ************************************************
%% quest(IntWord)
%% determinate if argument is an interrogative word
%% ************************************************
quest(who).
quest(what).
quest(how).
quest(do).
quest(did).
quest(does).
quest(when).
quest(where).
quest(can).
quest(would).
quest(could).
quest(are).
quest(is).
quest(am).
quest(have).
quest(has).
quest(which).
quest(should).
quest(why).

%% ************************************************
%% simpa(Sent,SimpSent)
%% simpa read Sent and try to find abbreviations.
%% Ex.sent=I'm ---- SimpSent=I am
%% ************************************************
simpa([],[]).
simpa([m|T],[am|T2])     :- simpa(T,T2),!.
simpa([ll|T],[will|T2])  :- simpa(T,T2),!.
simpa([d|T],[would|T2])  :- simpa(T,T2),!.
simpa([s|T],[is|T2])     :- simpa(T,T2),!.
simpa([H|T],[H|T2])      :- simpa(T,T2).

%% ************************************************
%% simp(Sent,SimpSent,X,Y)
%% simp read Sent and try to simplify it
%% ************************************************
simp([do,not|X],['don\'t'|Y],X,Y).
simp([can,not|X],[cant|Y],X,Y).
simp([cannot|X],[cant|Y],X,Y).
```

```
simp([will,not|X],[wont|Y],X,Y).
simp([dreamed|X],[dreamt|Y],X,Y).
simp([dreams|X],[dream|Y],X,Y).
simp([certainly|X],[yes|Y],X,Y).
simp([maybe|X],[perhaps|Y],X,Y).
simp([deutsch|X],[lang|Y],X,Y).
simp([dutch|X],[lang|Y],X,Y).
simp([francais|X],[lang|Y],X,Y).
simp([french|X],[lang|Y],X,Y).
simp([espanol|X],[lang|Y],X,Y).
simp([spanish|X],[lang|Y],X,Y).
simp([german|X],[lang|Y],X,Y).
simp([japanese|X],[lang|Y],X,Y).
simp([chinese|X],[lang|Y],X,Y).
simp([belgian|X],[lang|Y],X,Y).
simp([machine|X],[computer|Y],X,Y).
simp([machines|X],[computers|Y],X,Y).
simp([computers|X],[computer|Y],X,Y).
simp([am|X],[are|Y],X,Y).
simp([are|X],[am|Y],X,Y).
simp([your|X],[my|Y],X,Y).
simp([were|X],[was|Y],X,Y).
simp([me|X],[you|Y],X,Y).
simp([youre|X],[im|Y],X,Y).
simp([im|X],[youre|Y],X,Y).
simp([myself|X],[yourself|Y],X,Y).
simp([yourself|X],[myself|Y],X,Y).
simp([mum|X],[mother|Y],X,Y).
simp([mom|X],[mother|Y],X,Y).
simp([mammy|X],[mother|Y],X,Y).
simp([mother|X],[mammy|Y],X,Y).
simp([mommy|X],[mother|Y],X,Y).
simp([mummy|X],[mother|Y],X,Y).
simp([dad|X],[father|Y],X,Y).
simp([father|X],[dad|Y],X,Y).
simp([daddy|X],[father|Y],X,Y).
simp([you|X],[i|Y],X,Y).
simp([i|X],[you|Y],X,Y).
simp([my|X],[your|Y],X,Y).
simp([everybody|X],[everyone|Y],X,Y).
simp([nobody|X],[everyone|Y],X,Y).

simplify(List,Result) :- simp(List,Result,X,Y),!,simplify(X,Y).
simplify([W|Words],[W|NewWords]) :- simplify(Words,NewWords).
simplify([],[]).

%%  ************************************************
%%  lowercase(X,Y)
%%  Y  is the same string than X but in lower_case
%% ************************************************
low([],[]).
low([H|T],[H1|T1]) :- lower_case(H,H1),low(T,T1).
```

```
lower_case(X,Y) :- X >= 65,X=< 90,Y is X + 32,!.
lower_case(X,X).

% Clean_string Remove all punctuation in sentences
%% ************************************************
%% clean_string(Sent,NSent)
%% clean_string remove all punctiation is sentence, and put result in NSent
%% ************************************************
clean_string([C|Chars],[32|L]) :-
char_type(C,punctuation),clean_string(Chars,L).
clean_string([C|Chars],[C|L]) :-
not(char_type(C,punctuation)),clean_string(Chars,L).
clean_string([C|[]],[32]) :- char_type(C,punctuation),!.
clean_string([C|[]],[C]).

%Char_type - Determinate the type of a character
%% ************************************************
%% Char_type(X,Y)
%% Char_type determinates le type of a character
%% ************************************************
char_type(X,alphanumeric) :- X >= 65,X =< 90,!.
char_type(X,alphanumeric) :- X >= 97,X =< 123,!.
char_type(X,alphanumeric) :- X >= 48,X =< 57,!.
char_type(X,whitespace) :- X =< 32,!.
char_type(X,punctuation) :- X >= 33,X =< 47,!.
char_type(X,punctuation) :- X >= 58,X =< 63,!.  %% 64 est le code ascii de @
char_type(X,punctuation) :- X >= 91,X =< 96,!.
char_type(X,punctuation) :- X >= 123,X =< 126,!.
char_type(_,special).

%% ************************************************
%% remove_blanc(X,Y)
%% remove_blanc remove initial blanks of a string
%% ************************************************
remove_blanc([C|Chars],Result) :-
char_type(C,whitespace),!,remove_blanc(Chars,Result).
remove_blanc([C|Chars],[C|Chars]) :- not char_type(C,whitespace).


%% ************************************************
%% append(X,Y,Z)
%% append X to Y and result is in Z
%% ************************************************
append([],X,X).
append([H|T],B,[H|BT]) :- append(T,B,BT).


%% ************************************************
%% flatten
%% ************************************************
flatten(G,L) :- flatap(G,[],L).

list([]).
```

```
list([_|_]).

flatap([],X,X).
flatap(H,X,[H|X]) :- not(list(H)).
flatap([H|T],X,HTX) :- flatap(H,TX,HTX),flatap(T,X,TX).

%% ************************************************
%% append(X,Y,Z)
%% append X to Y and result is in Z
%% ************************************************
apb([],X,[X]).
apb([H|T],B,[H|BT]) :- apb(T,B,BT).

%% ************************************************
%% delete(X,Y,Z)
%% delete element X from L and result is in R
%% ************************************************
delete(X,[],[]).
delete(X,[X|T],T) :- !.
delete(X,[H|T],[H|R]) :- delete(X,T,R).

%% ************************************************************************
%% memb(X,Y,Z)
%% look if element X is in the list Y and Z will be at 0 if no and 1 if yes
%% ************************************************************************
memb(X,[],0).
memb(X,[X|T],1).
memb(X,[H|T],R) :- memb(X,T,R).

%% ************************************************
%% del_first_sp(L,NL) delete first elt of L if it's 32 (space blank)
%% L is a list of letters (ascii codes)
%% NL is a new list
%% ************************************************
del_first_sp([],[]).
del_first_sp([H],[H]).
del_first_sp([H|T],T) :- not(T=[]),(H=32).
del_first_sp([H|T],[H|T]) :- not(T=[]),not(H=32).

%% ************************************************
%% reverse(X,Y)
%% Y is the list X, but in the reverse order
%% ************************************************
reverse([],[]).
reverse([H|T],L) :- reverse(T,T2),append(T2,[H],L).

%% ************************************************
%% del_last(L,NL) get out last element of the list L. Result is in NL
%% L is a list
%% NL is the list L without his last elemeent
%% ************************************************
del_last([H],[]).
del_last([H|T],[H|T2]) :- del_last(T,T2).
```

```
%% ************************************************
%% check_num(L) checks if all elts of list L are numerics (ASCII CODES)
%% L is a list of ASCII codes
%% Fail is one ascii code is not a number
%% ************************************************
check_num([]).
check_num([H|T]) :- num(H),check_num(T).

%% ************************************************
%% del_spec(L,NL) get out the first @ elt in the list (@ = 64) L and result
is %% in NL
%% L is a list of ASCII codes
%% NL is the new list
%% ************************************************
del_spec([],[]).
del_spec([H|T],[H|T2]) :- not(H=64),del_spec(T,T2).
del_spec([H|T],NL) :- H=64,del_spec(T,NL).

%% ************************************************
%% aff_numjud(St) write in stream St number of judge
%% ************************************************
aff_numjud(St) :- judge(X),verif_num(X,St),write(St,X).

%% ************************************************
%% pr_jud(St,Com) print on Str the Com
%% Com is a commentary
%% Str is the stream
%% ************************************************
pr_jud(St,Com) :- int(true),!.
pr_jud(St,Com) :- name(Com,L),(L=[64,64,84,10]),!.
pr_jud(St,Com) :- name(Com,L),(L=[64,64,116,10]),!.
pr_jud(St,Com) :- name(Com,L),(L=[64,64,120,10]),!.
pr_jud(St,Com) :- name(Com,L),(L=[64,64,88,10]),!.
pr_jud(St,Com) :-
name(Com,L),del_spec(L,NL),del_last(NL,Lok),check_num(Lok),!.
pr_jud(St,Com) :-
write(St,'JUDGE'),aff_numjud(St),extr_time(H,M,S),write(St,'['),verif_num(H,S
t),write(St,H),write(St,':'),verif_num(M,St),write(St,M),write(St,':'),verif_
num(S,St),write(St,S),write(St,']'),write(St,Com).


%% ************************************************
%% r_input(I)
%% read characters up to EOL or ENTER
%% I is the input sentence
%% ************************************************
d_input(L,S,Str):- get0(H),apb(S,H,S2),di(H,L,S2,Str).

di(P,[],S,Str):- eof(P),!.
di(P,T,S,Str):- eol(P),dj(T,[32|S],Str),!.
            % anc. clausedi(P,[32|T]):- eol(P), dj(T),!.
di(P,[P|T],S,Str):- not(eol(P)),d_input(T,S,Str).

dj(L,S,Str):- get0(H), dk(H,L,S,Str).
```

```
dk(H,[],S1,Str):-
eof(H),del_first_sp(S1,S1ok),name(Sok,S1ok),pr_jud(Str,Sok),!.
dk(H,[],S1,Str):-
eol(H),del_first_sp(S1,S1ok),name(Sok,S1ok),pr_jud(Str,Sok),!.
dk(H,[32,H|L],S1,Str):- not(eol(H)),
not(eof(H)),del_first_sp(S1,S1ok),name(Sok,S1ok),pr_jud(Str,Sok),d_input(L,[H
],Str).

eof(-1).
eol(13).
eol(10).


%% ************************************************
%% extract_input(X,Y)
%% Extract words from X, transform it in ASCII codes and result is in Y
%% ************************************************
extract_input([],[]).
extract_input(InputC,[Word|InputD]) :-
InputC=[H|T],remove_blanc(InputC,X),extract_word(X,Y),del_1_word(X,Z),not(Y=[
]),name(Word,Y),extract_input(Z,InputD).
extract_input(InputC,InputD) :-
InputC=[H|T],remove_blanc(InputC,X),extract_word(X,Y),del_1_word(X,Z),Y=[],ex
tract_input(Z,InputD).

%% ************************************************
%% extract_words(X,Y)
%% extracts word every word of X and put it in Y
%% ************************************************
extract_word([],[]).
extract_word([H|T],[H|Y]) :- not(H=32),extract_word(T,Y).
extract_word([H|T],[]) :- (H=32).

%% ************************************************
%% del_1_word(X,Y) deletes the first word
%% X is a sentence
%% Y is the first word (up too 32=space)
%% ************************************************
del_1_word([],[]).
del_1_word([H|T],Z) :- not(H=32),del_1_word(T,Z).
del_1_word([H|T],T) :- H=32.

%% ************************************************
%% words not useful for our processing
%% ************************************************
nokey(you).
nokey(that).
nokey(this).
nokey(i).
nokey(blue).
nokey(black).
nokey(small).
nokey(big).
```

```
nokey(red).
nokey(pretty).
nokey(wonderful).
nokey(he).
nokey(she).
nokey(it).
nokey(we).
nokey(there).
nokey(here).
nokey(they).
nokey(of).
nokey(with).
nokey(at).
nokey(my).
nokey(many).
nokey(much).
nokey(your).
nokey(their).
nokey(above).
nokey(under).
nokey(beside).
nokey(next).
nokey(last).
nokey(front).
nokey(in).
nokey(on).
nokey(out).
nokey(beneath).
nokey(for).
nokey(the).
nokey(a).
nokey(an).
nokey(his).
nokey(her).
nokey(up).
nokey(by).
nokey(during).
nokey(however).
nokey(especially).
nokey(and).
nokey(right).
nokey(left).
nokey(top).
nokey(bottom).
nokey(then).
nokey(or).
nokey(as).
nokey(more).
nokey(which).
nokey(what).
nokey(who).
nokey(why).
```

```
nokey(where).
nokey(when).
nokey(how).
nokey(than).
nokey(all).
nokey(if).
nokey(about).

%% Find_key
find_key([],[]).
find_key([H|T],LKeys)       :- nokey(H),find_key(T,LKeys).
find_key([H|T],[H|LKeys])   :- not(nokey(H)),find_key(T,LKeys).


%%
************************************************************************
%% gener_he(Str,int) if int=0 we choose a randon number else we print each
%% time a comment like "I think..."
%% generate I think... randomly and print in the stream Str
%%
************************************************************************
gener_he(St,X) :- int(true),!.
gener_he(St,0) :- random(1,20,X),pr_he(St,X).
gener_he(St,1) :- random(1,10,X),pr_he(St,X).

pr_he(Str,X) :- X=1,write_comment('I think...',Str).
pr_he(Str,X) :- X=2,write_comment('That\'s interesting.',Str).
pr_he(Str,X) :- X=3,write_comment('Mmmmm...',Str).
pr_he(Str,X) :- X=4,write_comment('Wait a minute.',Str).
pr_he(Str,X) :- X=5,write_comment('It\'s difficult to find something to tell
about this.',Str).
pr_he(Str,X) :- X=6,write_comment('Wait a second, I think...',Str).
pr_he(Str,X) :- X=7,write_comment('Wait a moment I\'m very tired
today.',Str).
pr_he(Str,X) :- X=8,write_comment('Oh,oh...',Str).
pr_he(Str,X) :- X=9,write_comment('Ah,ah...',Str).
pr_he(Str,X) :- X=10,write_comment('Just a moment.',Str).
pr_he(Str,X) :- X>10.

%%
************************************************************************
%% disp_l(X) display list X
%%
************************************************************************
disp_l([]).
disp_l([H|T]) :- print(H),nl,disp_l(T).

%%
************************************************************************
%% memb(L,El) checks if El is an element of L.If no, memb(L,El) fails
%% L is a list
%% El is the element
%%
************************************************************************
```

```
memb([H|T],El) :- (H=El),!.
memb([H|T],El) :- not(H=El),memb(T,El).

%%
************************************************************************
%% ins_elt(List,Pos,Elem,NewList,Num) inserts element into a list
%% List is the initial list
%% Pos is the position where inserting new element
%% Elem is element to insert
%% NewList is the list with inserted element
%% Num is a work variable and allow to store the place where we are in the
%% list
%%
************************************************************************
ins_elt([],Pos,Elem,[],Num) :- !.
ins_elt([],Pos,Elem,NL,Num) :- !.
ins_elt([H|T],Pos,Elem,[H|T2],Num) :- Pos<Num,ins_elt(T,Pos,Elem,T2,Num),!.
ins_elt([H|T],Pos,Elem,[Elem|T2],Num)      :- Pos=Num,Numb is Num +
1,ins_elt([H|T],Pos,Elem,T2,Numb),!.
ins_elt([H|T],Pos,Elem,[H|T2],Num) :- not(Pos=Num),Numb is Num +
1,ins_elt(T,Pos,Elem,T2,Numb),!.

%%
************************************************************************
%% del_elt(List,Pos,NL,Num) delete the elt number Pos in the list List
%% List is a list of elements
%% Pos is the place where element must to be deleted
%% NL is the new list
%% Num is a work variable (number of element process now - need to be
%% initialized to 1)
%%
************************************************************************
del_elt([],Pos,[],Num) :- !.
del_elt([],Pos,NL,Num) :- !.
del_elt([H|T],Pos,[H|T2],Num) :- not(Pos=Num),Numb is
Num+1,del_elt(T,Pos,T2,Numb),!.
del_elt([H|T],Pos,T,Num) :- Pos=Num,Numb is Num + 1,del_elt(T,Pos,T,Numb).

%%
************************************************************************
%% transf_listb(Word,Res,BadTri,Resp) detects all bad tri-grams in word Word
%% and creates a pattern to search possible words. e.g. hovidays ->
%% [h,_,l,_,d,a,y,s] if ovi is an unknow tri-gram
%% Word is a list of letters
%% Res is a result
%% BadTri is the places of bad tri-grams e.g. [2,5,7]
%% Resp is the search pattern of the form [h,_,_,_,...]
%%
************************************************************************
transf_listb(_,L,[],Resp) :- Resp=L,!.
transf_listb([H|T],L,[Bt1|Bts],Resp) :-
transf_list([H|T],Res,Bt1,1),transf_listb(Res,Res,Bts,Resp).

%%
************************************************************************
```

```
%% transf_list(List,Listok,Num,NumElt) trasform the list(which is a search
%% pattern of the form [_,_,a,b...]
%% List is a initial search pattern
%% Listok is the new search pattern
%% Num is the place of bad tri-gram (e.g. 2 = place of context)
%% NumElt is the place of elt in the list process at this time
%%
************************************************************************
transf_list([],[],_,NumElt).
transf_list([H|T],['_'|T2],Num,NumElt) :- Tmp is Num-1,Tmp=NumElt,NumEltb is
NumElt + 1,transf_list(T,T2,Num,NumEltb),!.
transf_list([H|T],[H|T2],Num,NumElt)   :- Tmp is Num,Tmp=NumElt,NumEltb is
NumElt+1,transf_list(T,T2,Num,NumEltb),!.
transf_list([H|T],['_'|T2],Num,NumElt) :- Tmp is Num+1,Tmp=NumElt,NumEltb is
NumElt+1,transf_list(T,T2,Num,NumEltb),!.
transf_list([H|T],[H|T2],Num,NumElt)   :- NumEltb is NumElt +
1,transf_list(T,T2,Num,NumEltb),!.

%%
************************************************************************
%% transf_w(Word,ListLetters)
%% Word is a word
%% ListLetters is the list of letters of the word Word
%%
************************************************************************
transf_w(Word,ListLetters) :-
name(Word,WordAsc),build_list(WordAsc,ListLetters).

%%
************************************************************************
%% build_list(WordAsc,ListLetters)
%% WordAsc : list of codes ascii of letters of a word
%% ListLetters : list of letters corresponding to codes ascii
%%
************************************************************************
:- dynamic word/1.
build_list([],[]).
build_list([H|T],[L1|Ls]) :- name(L1,[H]),build_list(T,Ls).

take_word([]) :- not(word(X)).
take_word([X|Y]) :- word(X),retract((word(X))),take_word(Y).

gen_w(Bd4,Bd1) :-
l(X),rech_trig(X,*,Bd4,Tr,L),Tr=1,retract((l(X))),rempl_trig(X,L,Newlist),exi
st_word(Newlist,Bd1),gen_w(Bd4,Bd1).
gen_w(Bd4,Bd1) :-
l(X),rech_trig(X,*,Bd4,Tr,L),Tr=0,retract((l(X))),gen_w(Bd4,Bd1).
gen_w(Bd4,Bd1) :- not(l(X)).

rech_trig([X],P,Bd4,0,L) :- name(X,[Xasc]),not(Xasc=95).
rech_trig([X],P,Bd4,1,L) :- name(X,[Xasc]),Xasc=95,gen_trig([P,Var,*],L,Bd4).
rech_trig([X,Y|T],P,Bd4,Tr,L) :-
name(X,[Xasc]),not(Xasc=95),rech_trig([Y|T],X,Bd4,Tr,L).
rech_trig([X,Y|T],P,Bd4,1,L) :-
name(X,[Xasc]),Xasc=95,gen_trig([P,Var,Y],L,Bd4).
```

```
gen_trig([D,C,G],Ls,Bd4) :- db_findall(Bd4,tr(C,D,G),Ltr),cr_l_trig(Ltr,Ls).
gen_trig([D,C,G],Ls,Bd4) :- not(db_findall(Bd4,tr(C,D,G),Ltr)).

cr_l_trig([],[]).
cr_l_trig([H|T],[C|L]) :- assert((H)),tr(C,D,G),retract((H)),cr_l_trig(T,L).

rempl_trig(X,[],[]).
rempl_trig(X,[H|T],[Newlist|Y]) :- r_trig(X,H,Newlist),rempl_trig(X,T,Y).

r_trig([H|T],X,L) :-  name(H,[Xasc]),Xasc=95,append([X],T,L).
r_trig([H|T],X,[H|Y]) :- name(H,[Xasc]),not(Xasc=95),r_trig(T,X,Y).

exist_word([],Bd1).
exist_word([H|T],Bd1) :-
search_var(H,0),cr_word(H,L),name(Word,L),db_fetch(Bd1,s(_,_,Word,_,_),A),ass
ert((word(Word))),exist_word(T,Bd1).
exist_word([H|T],Bd1) :-
search_var(H,0),cr_word(H,L),name(Word,L),not(db_fetch(Bd1,s(_,_,Word,_,_),A)
),exist_word(T,Bd1).
exist_word([H|T],Bd1) :- search_var(H,1),assert((l(H))),exist_word(T,Bd1).

search_var([],0).
search_var([H|T],1) :- name(H,[Xasc]),Xasc=95.
search_var([H|T],Tr) :- name(H,[Xasc]),not(Xasc=95),search_var(T,Tr).

cr_word([],[]).
cr_word([H|T],[X|Y]) :- name(H,Asc),[X]=Asc,cr_word(T,Y).


%% ************************************************************************
%% maj_list(BadTri,BadTriMaj) reads list of bad trigrams and suppress some of
%% them.
%% e.g. BadTri = [2,3,4],we suppress 3 of the list because too closed of the
%% other ones
%% BadTri is a list of bad tri [2,4,5...]
%% BadTriMaj is the new list
%%
************************************************************************
maj_list([],[]).
maj_list([H1],[H1]).
maj_list([H1,H2|T],[H1|NL]) :- ((H2-H1) > 1),maj_list([H2|T],NL).
maj_list([H1,H2|T],NL)      :- not((H2-H1) > 1),maj_list([H1|T],NL).
maj_list([H1,H2],[H1,H2])   :- ((H2-H1) > 1).
maj_list([H1,H2],[H1])      :- not((H2-H1) > 1).

%%
************************************************************************
%% cons_arg(Word,BadTriList,Arg) builds an argument to search more similar
%% words in database
%% Word is a unknow word
%% BadTriList is a list of all bad tri-grams in the word Word
%% Arg is the search pattern of the form [a,b,c,_,_,_,g]
```

```
%%
************************************************************************
cons_arg(Word,BadTri,Arg) :-
transf_w(Word,ListLetters),maj_list(BadTri,BadTriok),transf_listb(ListLetters
,L,BadTriok,Arg).

%%
************************************************************************
%% gen_w(Patt,WordCor)
%% Patt : search pattern of the form [a,b,_,_,_,f,d]
%% WordCor : Possible corrected word
%%
************************************************************************
%gen_w([],WordCor).
%gen_w([H|T],WordCor) :-
H='_',T=[Conc,Right|Tb],db_fetch(Bd4,(X,_,Word,_,_),A)

%% ************************************************************************
%% gen_words_tri(Word,BadTriList,WordCor) generates words from Word by using
%% correct and false tri-grams
%% Word is the unknow word
%% BadTriList is a list of bad tri-grams for the word
%% WordCor is a list of possible correct word
%%
************************************************************************
gen_words_tri(Word,BadTriList,WordCor,Bd1,Bd4) :-
cons_arg(Word,BadTriList,Arg),Fact=..[l,Arg],assert((Fact)),gen_w(Bd4,Bd1),ta
ke_word(WordCor).

%%
************************************************************************
%% trans_l_asc(Ll,LAsc) transforms a list of letters into a list of ascii
%% codes
%%
************************************************************************
transf_l_asc([],[]).
transf_l_asc([H|T],[HAsc|T2]) :- name(H,[HAsc]),transf_l_asc(T,T2).

%% ************************************************************************
%% im_words(KeyWords,WordsCor,Bd1,Bd4,DebRes) take every keyword and checks
in
%% database wn_s if word exists
%% if words doesn't exists then we try to find similar words (with 1 or 2
%% letters changed
%% KeyWords is the set of keywords of the input sentence
%% WordsCor is a list of possible words for every keyword
%% Bdx are databases used
%% DebRes is begining of word
%%
************************************************************************
sim_words([],WordsCor,Bd1,Bd4,DebRes) :- (WordsCor=DebRes),!.
sim_words([H|T],WordsCor,Bd1,Bd4,DebRes) :-
w_exists(H,Bd1),sim_words(T,WordsCor,Bd1,Bd4,DebRes),!.
sim_words([H|T],WordsCor,Bd1,Bd4,DebRes) :-
not(w_exists(H,Bd1)),aj_code(H,NH),name(NH,NHAsc),bad_tr_l(NHAsc,BadTriList,B
```

```
d1,Bd4,1,R),not(BadTriList=[]),gen_words_tri(H,BadTriList,WordsCor,Bd1,Bd4),a
ppend(DebRes,WordsCor,DebResb),sim_words(T,WordsCor,Bd1,Bd4,DebResb).

%% ************************************************************************
%% add_w(DebRes,[H|OneWord],DebResb) add L to DebRes if OneWord is different
%% of []
%% ************************************************************************
add_w(Deb,X,[],Deb,Res).
add_w(Deb,X,Word,Deb,DebResb) :- append(Deb,[H|OneWord],DebResb).

%%
************************************************************************
%% search_w(Word,LWords) search for similars words of word Word (1 letter
%% changed....)
%% Word is the word not found
%% WordCor is a  similar words
%%
************************************************************************
search_w(Word,WordCor,Bd1,Bd4) :-
name(Word,WordAsc),bad_tri(WordAsc,WordCor,Bd1,Bd4,[]).

%% ************************************************************************
%% constr_fact(Letters,Fact) build a fact of type w([a,v,c|X])
%% Letters is a list of letters
%% Fact is the constructed fact
%%
************************************************************************
%constr_fact([H|T],Fact,Var) :- (NomFact=H),append([H|T],Var,Arg),Fact =..
[NomFact,Arg].

%%
************************************************************************
%% transf_w(WordList,WordAsc) transform a list of letters into a list of
ascii
%% codes
%% WordList is a list of letters
%% WordAsc is a list of ascii codes
%% ************************************************************************
transf_w([],[]) :- !.
transf_w([L1|Ls],[La1|LaS]) :- name(L1,[La1]),transf_w(Ls,LaS).

%%
************************************************************************
%% gen_words(Beg,End,WordCor) generates all possible words
%% Beg is the begining of words
%% End is a list of possible ends of words (list of lists...)
%% WordCor is a list of possible words
%%
************************************************************************
gen_words(Beg,[],WordCor,List) :- (WordCor=List).
gen_words(Beg,[H|T],WordCor,BegList) :-
append(Beg,H,WordList),transf_w(WordList,WordAsc),name(Word,WordAsc),apb(BegL
ist,Word,BegListNew),gen_words(Beg,T,WordCor,BegListNew).
```

```
%%
*************************************************************************
%% ver_fact([H|T]) verify if there is a word wich begin by H in memory
%%
*************************************************************************
ver_fact([]) :- fail.
ver_fact([H|T]) :- F=..[H,X],clause((F)),not(X=nofact).

%% *************************************************************************
%% bad_tri(Word,WordCor,Bd1,Bd4,Beg,Word) search for all positions of letters
%% which are not existing tri-grams
%% and try to find corrected words
%% Word is a word to check (list of ascii codes
%% WordCor is a word corrected by bad_tri (one by solution)
%% Bd1 is the database of WordNet words
%% Bd4 is the database of trigrams
%% Beg is the beginning of the word (part already processed - list of
letters)
%% Word is the complete word
%%
*************************************************************************
%bad_tri([H1|T],WordCor,Bd1,Bd4,Beg) :-
(T=[H2,H3|Tb]),name(L,[H1]),name(C,[H2]),name(R,[H3]),db_fetch(Bd4,tr(C,L,R),
A),apb(Beg,C,Begb),bad_tri(T,WordCor,Bd1,Bd4,Begb),!.
%bad_tri([H1|T],WordCor,Bd1,Bd4,Beg) :-
(T=[H2]),name(L,[H1]),name(C,[H2]),db_fetch(Bd4,tr(C,L,*),A),apb(Beg,C,Begb),
bad_tri(T,WordCor,Bd1,Bd4,Begb),!.
%bad_tri([H1],WordCor,Bd1,Bd4,Beg)  :- bad_tri([],WordCor,Bd1,Bd4,Beg),!.
%bad_tri([H1|T],WordCor,Bd1,Bd4,Beg) :-
(T=[H2,H3|Tb]),name(L,[H1]),name(C,[H2]),name(R,[H3]),not(db_fetch(Bd4,tr(C,L
,R),A)),ver_fact(Beg),constr_fact(Beg,F,Var),findall(Var,F,End),short_list(En
d,EndShort,300),gen_words(Beg,EndShort,WordCor,[]),!.
%bad_tri([H1|T],[],Bd1,Bd4,Beg) :-
(T=[H2,H3|Tb]),name(L,[H1]),name(C,[H2]),name(R,[H3]),not(db_fetch(Bd4,tr(C,L
,R),A)),not(ver_fact(Beg)),!.
%bad_tri([H1|T],[],Bd1,Bd4,Beg) :-
(T=[H2,H3|Tb]),name(L,[H1]),name(C,[H2]),name(R,[H3]),not(db_fetch(Bd4,tr(C,L
,R),A)),not(ver_fact(Beg)),!.
%bad_tri([],[],_,_,_).

%%
*************************************************************************
%% bad_tr_l(Word,BadTriList,Bd1,Bd4,Word) finds list of all bad tri-grams in
%% the word Word
%% Word is the word to check (list of ascii codes)
%% BadTriList is a list of bad trigrams (e.g. 2 = tri-gram with context like
2
%% letter doesn't exist in our database)
%% Bd1 is the database of words of WordNet (wn_s)
%% Bd4 is the database of tri-grams
%%
*************************************************************************
bad_tr_l([H1|T],L,Bd1,Bd4,Num,Resp) :-
T=[H2,H3|Tb],name(Left,[H1]),name(Conc,[H2]),name(Right,[H3]),db_fetch(Bd4,tr
(Conc,Left,Right),A),Numb is Num + 1,bad_tr_l(T,L,Bd1,Bd4,Numb,Resp),!.
```

```
bad_tr_1([H1|T],L,Bd1,Bd4,Num,Resp) :-
(T=[H2]),name(Left,[H1]),name(Conc,[H2]),db_fetch(Bd4,tr(Conc,Left,*),A),Numb
is Num + 1,bad_tr_1(T,L,Bd1,Bd4,Numb,Resp),!.
bad_tr_1([H1],L,Bd1,Bd4,Num,Resp) :- bad_tr_1([],L,Bd1,Bd4,Num,Resp),!.
bad_tr_1([H1|T],L,Bd1,Bd4,Num,Resp) :-
T=[H2,H3|Tb],name(Left,[H1]),name(Conc,[H2]),name(Right,[H3]),not(db_fetch(Bd
4,tr(Conc,Left,Right),A)),apb(Resp,Num,Respb),Numb is Num +
1,bad_tr_1(T,L,Bd1,Bd4,Numb,Respb).
bad_tr_1([H1|T],L,Bd1,Bd4,Num,Resp) :-
T=[H2],name(Left,[H1]),name(Conc,[H2]),not(db_fetch(Bd4,tr(Conc,Left,*),A)),a
pb(Resp,Num,Respb),Numb is Num + 1,bad_tr_1(T,L,Bd1,Bd4,Numb,Respb).
bad_tr_1([],L,_,_,_,Resp) :- Resp=L.

%% ***************************************************************************
%% word_exist(Words,UnkWords,Bd1) checks if Words exist in database and
%% filling UnkWords with words not found
%% Words is a list of words to search
%% UnkWords is a list of words not found in WordNet
%% Bd1 is the name of database to use
%%
***************************************************************************
word_exists([],[],_) :-!.
word_exists([H|T],[H|T2],Bd1) :-
not(db_fetch(Bd1,s(_,_,H,_,_),A)),word_exists(T,T2,Bd1),!.
word_exists([H|T],UW,Bd1) :-
db_fetch(Bd1,s(_,_,H,_,_),A),word_exists(T,UW,Bd1),!.

%%
***************************************************************************
%% w_exist(Word,Bd1) fail if word Word is not in WordNet
%% Word is a word
%% Bd1 is the database to use
%%
***************************************************************************
w_exists(Word,Bd1) :- db_fetch(Bd1,s(_,_,Word,_,_),A).

%%*************************************************************************
%% rempl_1(W,L,NW) replace the first letter of word Word by letter L and
%% return every solution
%% W is a word of the form a list of letters
%% L is the list of letters which replace the first letter
%% NW is the new word formed by replacement of one of this letters
%% !!!!!! RETURN LIKE LAST ELEMENT [] DO NOT FORGET TO GET OUT IT !!!
%% *************************************************************************
rempl_1([H|T],[L1|LS],[Lasc|T]) :- name(L1,Lb),(Lb=[Lasc|[]]).
rempl_1([H|T],[L1|LS],W)  :- not(LS=[]),rempl_1([H|T],LS,W).
rempl_1([H|T],[L1|LS],[]) :- (LS=[]).

%%
***************************************************************************
%% rempl_let(Word,Beg,NW,Bd3) replace every letter of word by a new letter in
%% LL,the new word formed by replacing
%% one letter is in NW (many solutions)
%% Word is the word to change
%% Beg is the begining of words (letters before letter to change) of the form
```

```
%% of a list
%% NW is the new word formed by replaceing a letter
%% Bd3 is the database which contains every key of keyboard and his closest
%% other keys
%%
************************************************************************
rempl_let([],Beg,[],Bd3).
rempl_let([H|T],Beg,NWok,Bd3) :-
name(L,[H]),db_findall(Bd3,pr([L,A,B,C,D,E,F,G]),Fact),search_l(Fact,Letters)
,rempl_l([H|T],Letters,NW),not(NW=[]),append(Beg,NW,NWok),name(Temp,NWok).
rempl_let([H|T],Beg,NWok,Bd3) :- apb(Beg,H,Begb),rempl_let(T,Begb,NWok,Bd3).

%%
************************************************************************
%% search_l(Fact,Letters) take a fact of the form pr([a,b,c,d,e,f,g,h]),and
%% put the list of letters in Letters
%% Fact is a list of 1 elt which contains pr([a,b,c,d,e,f,g,h]).
%% Letters are letters of the fact (a list)
%% ************************************************************************
search_l([Fact],Letters) :- assert((Fact)),pr(Letters),retract((Fact)).

%% ************************************************************************
%% cgt_l(Code,Words,Bd1,Bd2,Bd3) change every letter of the word by keyboards
%% proximity and
%% try to find correct words (present in WordNet using keyboard proximity
%% Code is the ascii code of word to use.
%% Bd3 is the database with keyboard proximity - every entry is of the form
%% pr([a,q,w,s,x,z,' ',' ']) - every time 8 elts
%% Words is a list of words presents in our subjects found by changing a
%% letter from initial word
%%
************************************************************************
cgt_l([],[],Bd1,Bd2,Bd3).
cgt_l([H|T],NewWord,Bd1,Bd2,Bd3) :-
rempl_let([H|T],[],NewWordasc,Bd3),not(NewWordasc=[]),name(NewWord,NewWordasc
).

%% ************************************************************************
%% cor_spel(UnkWords,Words,Bd3) verify every word of the sentence and try to
%% find words present in WordNet
%% UnkWords is a list of unknowed words in WordNet
%% Words are found by taking every word of the Sent,and try to replace a
%% letter up to find a correct word
%% Bd1-Bd3 are usefu Bd's
%%
************************************************************************
cor_spel([],[],Bd1,Bd2,Bd3).
cor_spel([H|T],Words,Bd1,Bd2,Bd3) :-
name(H,Code),cgt_l(Code,Words,Bd1,Bd2,Bd3).
cor_spel([H|T],Words,Bd1,Bd2,Bd3) :- cor_spel(T,Words,Bd1,Bd2,Bd3).

%%
************************************************************************
%% verif_t(Word,Bd4) verify if all sequences of letters of words respect
%% trigrams
```

```
%% Word is a word in the form of a list (asc code)
%%
**************************************************************************
verif_t([],Bd4).
verif_t([H1|T],Bd4) :-
(T=[H2,H3|Tb]),name(Left,[H1]),name(Conc,[H2]),name(Right,[H3]),db_fetch(Bd4,
tr(Conc,Left,Right),A),verif_t(T,Bd4),!.
verif_t([H1|T],Bd4) :-
(T=[H2,H3]),name(Left,[H1]),name(Conc,[H2]),name(Right,[H3]),db_fetch(Bd4,tr(
Conc,Left,Right),A),verif_t(T,Bd4),!.
verif_t([H1|T],Bd4) :-
(T=[H2]),name(Left,[H1]),name(Conc,[H2]),db_fetch(Bd4,tr(Conc,Left,*),A),veri
f_t(T,Bd4),!.
verif_t([H1|T],Bd4) :-
(T=[]),name(Conc,[H1]),db_fetch(Bd4,tr(H1ok,*,*),A),verif_t(T,Bd4),!.


%%
**************************************************************************
%% ver_trig(Word,Bd4) verify if all sequences of letters in words are used in
%% trigrams of letters
%% if no,then we suppose word not correct
%% !!! every word is delimited by $ before and after e.g. $hello$
%%
**************************************************************************
ver_tri(vide,Bd4) :- !.
ver_tri(Word,Bd4) :- name(Word,[H|T]),verif_t([H|T],Bd4).

%%
**************************************************************************
%% ins_first(List,NewList) add to the list List * in first position
%%
**************************************************************************
ins_first([],[X]).
ins_first([H|T],[42|T2]) :- (T2=[H|T]).

%%
**************************************************************************
%% aj_code(Word,NewWord) add * to the begining and the end of the word e.g.
%% $hello$
%% Word is the word to process
%% Newword is the new word after appending control characters
%%
**************************************************************************
aj_code([],vide) :- !.
aj_code(Word,NewWord) :-
name(Word,WordAsc),apb(WordAsc,42,Wordtmp),ins_first(Wordtmp,NewWordasc),name
(NewWord,NewWordasc).

%%
**************************************************************************
%% short_list(L,NL,Cpt) takes Cpt first elements of L.Results are in NL
%% L is a list
%% NL is a list formed with the Cpt first elts of NL
```

```
%%
*************************************************************************
short_list([],[],_).
short_list([H|T],[H|T2],Cpt) :- Cpt>0,Cpt2 is Cpt-1,short_list(T,T2,Cpt2).
short_list([H|T],[],Cpt) :- (Cpt=0).

%%
*************************************************************************
%% cal_dist(W,L,WCor) calculates distance between word W and a list of
%% possible words
%% W is a word not found in WordNet
%% L is a list of possible words
%% WCor is the list less distant word of W
%%
*************************************************************************
cal_dist(W,[],[]).
cal_dist(W,[H|T],[H|T2]) :-
name(W,Wasc),name(H,Hasc),length(Wasc,Lw),length(Hasc,Lh),(Lw>=Lh),(Lw-
Lh<3),cal_dist(W,T,T2).
cal_dist(W,[H|T],[H|T2]) :-
name(W,Wasc),name(H,Hasc),length(Wasc,Lw),length(Hasc,Lh),(Lw<Lh),(Lh-
Lw<3),cal_dist(W,T,T2).
cal_dist(W,[H|T],X) :-
name(W,Wasc),name(H,Hasc),length(Wasc,Lw),length(Hasc,Lh),(Lw>Lh),not(Lw-
Lh<3),cal_dist(W,T,X).
cal_dist(W,[H|T],X) :-
name(W,Wasc),name(H,Hasc),length(Wasc,Lw),length(Hasc,Lh),(Lw<Lh),not(Lh-
Lw<3),cal_dist(W,T,X).

%%
*************************************************************************
%% genw(UnkWord,Word,Beg,Asc) generates words by inserting letter
%% UnkWord is ascii code of unknow word
%% Word is a correct word (1 by solution)
%% Beg is the begining of word already process
%% Asc is ascii code to add
%% ***********************************************************************
genw([],[],Beg,CodeAsc) :- !.
genw([H|T],Word,Beg,CodeAsc) :-
not(CodeAsc>122),EndWord=[H,CodeAsc|T],append(Beg,EndWord,Word).
genw([H|T],Word,Beg,CodeAsc) :- not(CodeAsc>122),CodeAscb is
CodeAsc+1,genw([H|T],Word,Beg,CodeAscb).
genw([H|T],Word,Beg,CodeAsc) :-
(CodeAsc>122),apb(Beg,H,Begb),genw(T,Word,Begb,97).

%%
*************************************************************************
%% genw1(UnkWord,Word,Beg,Asc) generates words by inserting letter in first
%% position
%% UnkWord is ascii code of unknow word
%% Word is a correct word (1 by solution)
%% Beg is the begining of word already process
%% Asc is ascii code to add
%%
*************************************************************************
```

```
genw1([],[],Beg,CodeAsc) :- !.
genw1([H|T],Word,Beg,CodeAsc) :- not(CodeAsc>122),Word=[CodeAsc,H|T].
genw1([H|T],Word,Beg,CodeAsc) :- not(CodeAsc>122),CodeAscb is
CodeAsc+1,genw1([H|T],Word,Beg,CodeAscb).
genw1([H|T],[],Beg,CodeAsc) :- (CodeAsc>122).

%% ************************************************************************
%% genwdel(UnkWord,Word) generates all words from UnkWord by suppressing
every
%% letter (one at a time) e.g. housre -> house
%% UnkWord is a list of Unknow words
%% Word is a correct word (1 by solution)
%%
************************************************************************
genwdel([],[],Beg) :- !.
genwdel([H|T],Word,Beg) :- append(Beg,T,Word).
genwdel([H|T],Word,Beg) :- apb(Beg,H,Begb),genwdel(T,Word,Begb).

%% ************************************************************************
%% genw_ins(H,Words) generates all words possible by inserting a letter (e.g.
%% hose -> [horse,house..])
%% H : Unknow word
%% Word : Possible correct words found
%%
************************************************************************
genw_ins(H,Word) :- name(H,WAsc),genw(WAsc,Word,[],97).

%%
************************************************************************
%% genw_ins1(H,Words) generates all words possible by inserting a letter in
%% first position in a word
%% H : Unknow word
%% Word : Possible correct words found
%% ************************************************************************
genw_ins1(H,Word) :- name(H,WAsc),genw1(WAsc,Word,[],97).

%%
************************************************************************
%% genw_del(H,Words) generates all words possible by deleting a letter in
%% words(e.g. housre -> [...house...])
%% H : Unknow word
%% Word : Possible correct words found
%%
************************************************************************
genw_del(H,Word) :- name(H,WAsc),genwdel(WAsc,Word,[]).

%%
************************************************************************
%% check_insdel(UnkWords,NewWord,Bd1,Bd2,Bd3)
%% UnkWords : List of unknows words in sentence
%% NewWords : Correct word found by searching by insertion/deletion of
letters
%% one words by solution)
%%
************************************************************************
```

```
check_insdel([],[]).
check_insdel([H|T],NewWord) :-
genw_ins1(H,NewWordAsc),name(NewWord,NewWordAsc).
check_insdel([H|T],NewWord) :-
genw_ins(H,NewWordAsc),name(NewWord,NewWordAsc).
check_insdel([H|T],NewWord) :-
genw_del(H,NewWordAsc),name(NewWord,NewWordAsc).

%%
*************************************************************************
%% search_sub(Word,Sub) search for if word is a subject,if no try to find the
%% singular of the word and check again
%% Word is a word
%% Sub is the found subject or nosub
%%
*************************************************************************
search_sub(Word,Sub) :- if_subj([Word],Sub),not(Sub=nosub),!.
search_sub(Word,Sub) :-
if_subj([Word],SubTmp),SubTmp=nosub,plural_sub(Word,WordSing),if_subj([WordSi
ng],Sub).

%% *************************************************************************
%% make_comment(Keys,Sent,KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,St) generates a
%% commentary
%% Sub : subjects available
%% Nosub : words no subjects
%% Sent : sentence already simplify
%% KeyWords : useful words of sentence
%% Bd1,Bd2,Bd3,Bd4 : databases
%% Comment : Commentary to be printed
%% St : File
%%
*************************************************************************
make_comment(Sub,Nosub,[X],KW,Bd1,Bd2,Bd3,Bd4,Comment,St) :-
setupmode(true),name(X,Xasc),Xasc=[64,64,88],(Comment=' '),!.

%% interrogation mode
make_comment(Sub,Nosub,KW,Bd1,Bd2,Bd3,Bd4,Comment,St) :-
int(true),(Comment=''),!.

%% We do a change of judge - no subject no keywords
make_comment([],Nosub,[X],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,St) :-
name(X,Xasc),Xasc=[64,64,116],(Comment=''),!.

%% User wants go away
make_comment([],Nosub,[X],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,St) :-
name(X,Xasc),Xasc=[64,64,119,111,117,116],(Comment=''),!.

%% we do a change of judge
make_comment([],Nosub,[X],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,St) :-
name(X,Xasc),Xasc=[64,64,110,110],(Comment=''),!.

%% for sentence predefined
make_comment(Sub,Nosub,Sentence,KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,St) :-
predef(Sentence,Comment),write_comment(Comment,St),!.
```

```
%% We have a subject and this a question
make_comment(Sub,Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,
St) :-
quest(FirstWord),not(Sub=[]),make_comment_quest(Sub,Nosub,[FirstWord|Sentence
],Comment,St),!.

%% we have not subject and this is a question - using keyb proximity to try
to %% find subject
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
quest(FirstWord),word_exists(KeyWords,UnkWords,Bd1),plur_sing(UnkWords,UnkWor
dsb),not(UnkWords=[]),cor_spel(UnkWordsb,NewWord,Bd1,Bd2,Bd3),aj_code(NewWord
,NewWordC),ver_tri(NewWordC,Bd4),if_subj([NewWord],Sub),not(Sub=nosub),make_c
omment_aff([Sub],Nosub,[FirstWord|Sentence],Comment,St),!.

%% NEW we have not a subject and this is a question - check if user
%% insert/delete a letter(typing error) in words to find a subject
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
quest(FirstWord),word_exists(KeyWords,UnkWords,Bd1),plur_sing(UnkWords,UnkWor
dsb),not(UnkWordsb=[]),check_insdel(UnkWordsb,NewWord),search_sub(NewWord,Sub
),not(Sub=nosub),make_comment_aff([[NewWord,Sub]],[FirstWord|Sentence],Commen
t,St),!.

%% NEW
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
quest(FirstWord),sim_words(KeyWords,WordsCor,Bd1,Bd4,[]),if_subj(WordsCor,Sub
),not(Sub=nosub),make_comment_aff([Sub],Nosub,[FirstWord|Sentence],Comment,St
),!.

%% We have not a subject and this a question
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
quest(FirstWord),make_comment_quest([],Nosub,[FirstWord|Sentence],Comment,St)
,!.

%% We have a subject and a affirmation
make_comment(Sub,Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,
St) :-
not(quest(FirstWord)),not(Sub=[]),make_comment_aff(Sub,Nosub,[FirstWord|Sente
nce],Comment,St),!.

%% We have not a subject and it's an affirmation - using keyboard proximity
to
%% find some words
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
not(quest(FirstWord)),word_exists(KeyWords,UnkWords,Bd1),plur_sing(UnkWords,U
nkWordsb),not(UnkWords=[]),cor_spel(UnkWordsb,NewWord,Bd1,Bd2,Bd3),aj_code(Ne
wWord,NewWordC),ver_tri(NewWordC,Bd4),if_subj([NewWord],Sub),not(Sub=nosub),m
ake_comment_aff([Sub],Nosub,[FirstWord|Sentence],Comment,St),!.

%% NEW we have not a subject and this is an affirmation - check if user
```

```
%% insert/delete a letter in words to finda subject
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
not(quest(FirstWord)),word_exists(KeyWords,UnkWords,Bd1),plur_sing(UnkWords,U
nkWordsb),not(UnkWordsb=[]),check_insdel(UnkWordsb,NewWord),search_sub(NewWor
d,Sub),not(Sub=nosub),make_comment_aff([[NewWord,Sub]],[FirstWord|Sentence],C
omment,St),!.

%% no subject - nothing found with keyboard proximity correct spelling
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
not(quest(FirstWord)),sim_words(KeyWords,WordsCor,Bd1,Bd4,[]),short_list(Word
sCor,WordsCorShort,200),plur_sing(WordsCorShort,[H|T]),cal_dist(H,T,Ld),if_su
bj(Ld,Sub),not(Sub=nosub),make_comment_aff([Sub],Nosub,[FirstWord|Sentence],C
omment,St),!.

%% We have not a subject and a affirmation
make_comment([],Nosub,[FirstWord|Sentence],KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,S
t) :-
not(quest(FirstWord)),make_comment_aff([],Nosub,[FirstWord|Sentence],Comment,
St),!.

%% ************************************************************************
%% make_comment_aff(Sub,Nosub,Sent,Keywords,Comment,St) generates a
commentary
%% for affirmation with and without subjects
%%
************************************************************************
%% We have a subject and a affirmation
make_comment_aff(Sub,Nosub,[FirstWord|Sentence],Comment,St) :-
not(Sub=[]),research_comment_sub(FirstWord,Sub,Lclause1,aff),research_comment
_nosub(Nosub,Lclause1,aff),take_best_comment(Comment,aff),write_comment(Comme
nt,St),[H1|H2]=Sub,trt_red_quest(H1,St),!.

%% We have not a subject and a affirmation
make_comment_aff([],Nosub,[FirstWord|Sentence],Comment,St) :-
random(1,10,R),tabtostring([FirstWord|Sentence],R,Com),write_comment(Com,St),
gener_com_sec(NumClause,Comment),write_comment(Comment,St),Q=gener_com_sec,P=
..[Q,NumClause,Comment],retract((P)),assertz((P)),!.


%% ************************************************************************
%% make_comment_quest(Keys,Sent,Comment) generates a commentary for
%% affirmation with and without subjects
%%
************************************************************************
%% We have a subject and a question
make_comment_quest(Sub,Nosub,[FirstWord|Sentence],Comment,St) :-
not(Sub=[]),research_comment_sub(FirstWord,Sub,Lclause1,quest),research_comme
nt_nosub(Nosub,Lclause1,quest),take_best_comment(Comment,quest),write_comment
(Comment,St),[H1|H2]=Sub,trt_red_quest(H1,St),!.

% We haven't subject and it's a question
```

```
make_comment_quest([],Nosub,[FirstWord|Sentence],Comment,St) :-
gener_sec(NumClause,Comment),write_comment(Comment,St),Q=gener_sec,P=..[Q,Num
Clause,Comment],retract((P)),assertz((P)),!.


%% ************************************************************************
%% Fonction to display a red_quest about a subject or another if this subject
%% doesn't exist in our list
%% it needs subject, file
%% ************************************************************************
trt_red_quest(Sub,St) :-
red_quest(Sub,_,Com),write_comment(Com,St),retract(red_quest(Sub,_,Com)),!.
trt_red_quest(Sub,St) :-
not(red_quest(Sub,_,Com)),red_quest(vide,N,Com2),write_comment(Com2,St),pgm_s
leep(N),retract(red_quest(vide,N,Com2)),!.

%% ************************************************************
%% Fonction to sleep if the comment is "...going to toilets" ...
%% ************************************************************
pgm_sleep(X) :- X=dormir,delay(20000000).
pgm_sleep(X) :- not(X=dormir).


%% ************************************
%% Fonction  to translate a list in string
%% ************************************
tabtostring([],R,[]).
tabtostring(X,R,S) :- R>5,tabtoascii(X,[],A),name(S,A).
tabtostring(X,R,S) :- not(R>7),S=''.

tabtoascii([],T,Res) :- Res=T.
tabtoascii([H|T],Temp,Res) :-
name(H,X),append(Temp,X,V),append(V,[32],W),tabtoascii(T,W,Res).

%% ***************************************************
%% Fonctions to write Elizabeth's responses
%% ***************************************************
:- dynamic cutl/1.
%% General function used to display comments
write_comment(C,St) :-
make_delay(lot),name(C,T),put_pos(0),!,insert_simul(T,Cnew,Comfi),!,flatten(C
omfi,Listefi),ctrl_fin(Listefi,NListe),write_fi(NListe,[],St),flatten(Cnew,Li
st),write_gen(List),!.

%% ***************************************************
%% ctrl_fin adds ascii code 10(ENTER) to the end if needed
%% ***************************************************
ctrl_fin([],[]).
ctrl_fin([H|T],[H|T]) :- last_elm([H|T],Elm),Elm=10.
ctrl_fin([H|T],L) :- last_elm([H|T],Elm),not(Elm=10),append([H|T],[10],L).

%% ***************************************************
%% write_gen displays comment doing errors and displaying backspaces...
%% ***************************************************
```

```
write_gen([]).
write_gen([H]) :- name(X,[H]),print(X),nl.
write_gen([8,32,8|T]) :-
name(X,[8]),make_delay(little),print(X),name(Y,[32]),print(Y),print(X),write_
gen(T).
write_gen([H|T]) :-
not(T=[]),not(H=32),make_delay(little),name(X,[H]),print(X),write_gen(T).
write_gen([H|T]) :-
not(T=[]),H=32,make_delay(middle),name(X,[H]),print(X),write_gen(T).

%% *****************************************************
%% give_pos gives the positioon where we are to scan comment in the list
%% *****************************************************
give_pos(Pos) :- cutl(Pos),!.

%% *****************************************************
%% add_pos adds a new position
%% *****************************************************
add_pos(Addpos) :- cutl(Pos),Newpos is Pos +
Addpos,Q=cutl,P=..[Q,Pos],retract((P)),N=..[Q,Newpos],assert((N)).

%% *****************************************************
%% function which put a new position
%% *****************************************************
put_pos(Newpos) :-
Q=cutl,P=..[Q,0],not(clause((P))),N=..[Q,Newpos],assert((N)).
put_pos(Newpos) :-
Q=cutl,P=..[Q,0],clause((P)),retract((P)),N=..[Q,Newpos],assert((N)).

%% *****************************************************
%% insert_simul manages insertion of line feed and spelling errors
%% *****************************************************
insert_simul([],[],[]).
insert_simul([H|T],[L|Y],[Nline|Rest]) :-
random(48,55,Cutline),retractall(cutl(A)),put_pos(0),insert_nl([H|T],Cutline,
Llu,Nline),del_list(Llu,[H|T],Lstay),random(1,50,Z),insert_mist(Nline,Z,L),in
sert_simul(Lstay,Y,Rest).

%% *****************************************************
%% delete a list from a list
%% *****************************************************
del_list([],L,L).
del_list([H|T],L,Res) :- delete(H,L,Nl),del_list(T,Nl,Res).

%% *****************************************************
%% fonction which insert line feed
%% *****************************************************
insert_nl([],_,[],[]).
insert_nl([H|T],Cutline,[H|Y1],[H|Y2]) :-
H=32,give_pos(Pos),Pos=<Cutline,add_pos(1),insert_nl(T,Cutline,Y1,Y2).
insert_nl([H|T],Cutline,[H],[10]) :- H=32,give_pos(Pos),Pos>Cutline.
insert_nl([H|T],Cutline,[H|Y1],[H|Y2]) :-
not(H=32),add_pos(1),insert_nl(T,Cutline,Y1,Y2).
```

```
%% ******************************************************
%% insert_mist inserts in a list ascii codes of 3 types of errors
%% spelling errors + backspace to correct it
%% spelling error with keyboard proximity
%% spelling error + backspaces to correct them (more than 1 backspace)
%% ******************************************************
insert_mist([],Z,[]).
insert_mist([H|T],Z,[Car,8,H|Y]) :-
not(T=[]),H>96,H<123,Z=10,!,mist_back(H,Car),random(1,50,Nn),insert_mist(T,Nn
,Y).
insert_mist([H|T],Z,[Car,8,H|Y]) :-
not(T=[]),H>96,H<123,Z=15,!,mist_prox(H,Car),random(1,50,Nn),insert_mist(T,Nn
,Y).
insert_mist([H|T],Z,[L|Y]) :-
not(T=[]),H>96,H<123,Z=20,!,random(4,20,B),mist_back_more(B,H,T,L),random(1,5
0,Nn),insert_mist(T,Nn,Y).
insert_mist([H|T],Z,[H|Y]) :- random(1,50,Nn),insert_mist(T,Nn,Y).

%% ******************************************************
%% function which generates an error
%% ******************************************************
mist_back(H,Car) :- H>98,H<123,random(97,H,Car).
mist_back(H,Car) :- H=122,random(97,121,Car).
mist_back(H,Car) :- H=<98,random(99,122,Car).

%% ******************************************************
%% function which generates an error with keyboard proximity
%% ******************************************************
mist_prox(H,Car) :-
name(Hr,[H]),prox(Numclause,Hr,C),name(C,T),T=[Car],Q=prox,P=..[Q,Numclause,H
r,C],retract((P :- ! )),assertz((P :-! )).

%% fonction qui fait une faute de touche avec plusieurs backspaces

%% ******************************************************
%% function which make a spelling error with few backspaces to correct it
%% ******************************************************
mist_back_more(B,Car,T,L) :- length(T,Le),B=<Le,add_backs(Car,T,B,L).
mist_back_more(B,Car,T,[Car]) :- length(T,Le),B>Le.

%% ******************************************************
%% creates part of list with error and backspaces and inserts it in the
%% principal comment
%% ******************************************************
add_backs(Car,T,B,L) :-
B=<7,mist_back(Car,Ncar),append([Ncar],T,Nt),car_list(Nt,8,B,L1,Lt),append(L1
,Lt,L2),append(L2,[Car],L).
add_backs(Car,T,B,L) :-
B>7,B=<14,mist_prox(Car,Ncar),append([Ncar],T,Nt),car_list(Nt,8,B,L1,Lt),appe
nd(L1,Lt,L2),append(L2,[Car],L).
add_backs(Car,T,B,[Car]) :- B>14,B<21.

%% fonction qui prend en argument une liste initiale, le caractere a placer,
%% le nombre a mettre de
```

```
%% ces caracteres et rend deux listes Exemple [8,32,8] represente un
backspace
%% le 70 est une faute car ca devait etre 65
%% car_list([70,66,67,68,69],8,3,L,P) donne
%% L=[70,66,67] et P=[8,32,8,8,32,8,8,32,8]
car_list([H|T],C,0,[],[]).
car_list([H|T],C,Nb,[H|Hy],[C,32,C|Y]) :- not(Nb=0),N is Nb -
1,car_list(T,C,N,Hy,Y).

%% ************************************************************************
%% Fonction pour tranformer une chaine de caracteres en tableaux de mots
%%************************************************************************
trans_tab(X,T) :- name(X,Y),cons_tab(Y,[],T).

cons_tab([],Nword,[Word]) :- (H=32),name(Word,Nword).
cons_tab([H|T],Z,Y) :- not(H=32),append(Z,[H],R),cons_tab(T,R,Y).
cons_tab([H|T],Nword,[Word|B]) :- (H=32),name(Word,Nword),cons_tab(T,[],B).

%% *********************************************
%% Fonction pour imiter humain au clavier (delay)
%%*********************************************
make_delay(little) :- random(100000,200000,N),delay(N).
make_delay(middle) :- random(200000,400000,N),delay(N).
make_delay(lot) :- random(400000,800000,N),delay(N).

delay(X) :- sleep_p(X).

%% ***************************************
%% search_sub_verb searchs verbs after nouns
%% ***************************************
search_sub_verb([],[]).
search_sub_verb([H|T],[Hr|Z]) :- sub_verb(H,Hr),search_sub_verb(T,Z).
search_sub_verb([H|T],Z) :- not(sub_verb(H,Hr)),search_sub_verb(T,Z).


repl([],X,[]).
repl([H|T],Syn,[Y|Com]) :- s_syn(H,Syn,Y), !, repl(T,Syn,Com).
repl([H|T],Syn,[H|Com]) :- repl(T,Syn,Com).

s_syn(X,[[X,N]|T],N) :- !.
s_syn(X,[[A,N]|T],Y) :- s_syn(X,T,Y).

%% search_syn takes a list of words and call parallel process
search_syn([predef],_,_,Llsub,Llsub,St,Bd1,Bd2).
search_syn(A,[],_,Llsub,Llsub,St,Bd1,Bd2) :- not(A=[predef]).
search_syn(C,[H|T],X,Llsub,Llsub,St,Bd1,Bd2) :- not(C=[predef]),X>=1.
search_syn(C,[H|T],X,_,Syn,St,Bd1,Bd2) :-
not(C=[predef]),X<1,findall(Hkeys,paral(St,[H|T],Skeys,Hkeys,Bd1,Bd2),Lkeys),
[Syn|B]=Lkeys.

%% find_syn   Find a synonymous that is in our subjects
find_syn([],[],_,_).

% word doesn't exist in the dictionary
```

```
find_syn([H|T],T1,Bd1,Bd2) :-
not(db_fetch(Bd1,s(X,_,H,_,_),A)),find_syn(T,T1,Bd1,Bd2).

% search synonyms in two senses
find_syn([H|T],[[H,Sub]|T1],Bd1,Bd2) :-
db_fetch(Bd1,s(X,_,H,_,_),A),db_findall(Bd2,hyp(X,Y),L),not(L=[]),write_cl(L)
,search_hyp(X,L1,Bd1),db_findall(Bd2,hyp(Y2,X),Lx),write_cl(Lx),search_hyp2(X
,L2,Bd1),append(L1,L2,Lok),!,if_subj(Lok,Sub),find_syn(T,T1,Bd1,Bd2).

% search synomyms in sense 1
find_syn([H|T],[[H,Sub]|T1],Bd1,Bd2) :-
db_fetch(Bd1,s(X,_,H,_,_),A),db_findall(Bd2,hyp(X,Y),L),L=[],db_findall(Bd2,h
yp(Y2,X),Lx),not(Lx=[]),write_cl(Lx),search_hyp2(X,L2,Bd1),!,if_subj(L2,Sub),
find_syn(T,T1,Bd1,Bd2).

% find no synonyms
find_syn([H|T],[[H,nosub]|T1],Bd1,Bd2) :-
db_fetch(Bd1,s(X,_,H,_,_),A),db_findall(Bd2,hyp(X,Y),L),L=[],db_findall(Bd2,h
yp(Y2,X),Lx),Lx=[],!,find_syn(T,T1,Bd1,Bd2).

% search synomyms in sense 2
find_syn([H|T],[[H,Sub]|T1],Bd1,Bd2) :-
db_fetch(Bd1,s(X,_,H,_,_),A),db_findall(Bd2,hyp(X,Y),L),not(L=[]),write_cl(L)
,search_hyp(X,L1,Bd1),db_findall(Bd2,hyp(Y2,X),Lx),Lx=[],!,if_subj(L1,Sub),fi
nd_syn(T,T1,Bd1,Bd2).

%% find_hyp   Find a hypernym that is in our subjects
find_hyp([],[],_,_).
find_hyp([H|T],Y,Bd1,Bd2) :-
not(db_fetch(Bd1,s(X,_,H,_,_),A)),find_hyp(T,Y,Bd1,Bd2).
find_hyp([H|T],[[H,Sub]|T1],Bd1,Bd2) :-
db_fetch(Bd1,s(X,_,H,_,_),A),db_findall(Bd2,hyp(X,Y),L),not(L=[]),[Hl|Tl]=L,a
ssert(Hl),hyp(X,Y),!,research_hyp(Y,Sub,Bd1,Bd2,1),find_hyp(T,T1,Bd1,Bd2).
find_hyp([H|T],[[H,nosub]|T1],Bd1,Bd2) :-
db_fetch(Bd1,s(X,_,H,_,_),A),db_findall(Bd2,hyp(X,Y),L),L=[],!,find_hyp(T,T1,
Bd1,Bd2).

research_hyp(Codex,Sub,Bd1,Bd2,Num) :- (Num <
10),db_findall(Bd2,hyp(Codex,Codey),L),not(L=[]),[Hl|Tl]=L,assert(Hl),hyp(Cod
ex,Codey),search_hyp(Codex,L1,Bd1),if_subj(L1,Rsub),Rsub=nosub,Nnum is Num +
1,research_hyp(Codey,Sub,Bd1,Bd2,Nnum).
research_hyp(Codex,Sub,Bd1,Bd2,Num) :- (Num <
10),db_findall(Bd2,hyp(Codex,Codey),L),not(L=[]),[Hl|Tl]=L,assert(Hl),search_
hyp(Codex,L1,Bd1),if_subj(L1,Sub),not(Sub=nosub).
research_hyp(Codex,Sub,Bd1,Bd2,Num) :- (Num <
10),db_findall(Bd2,hyp(Codex,Codey),L),L=[],Sub=nosub.
research_hyp(Codex,Sub,Bd1,Bd2,10) :- Sub=nosub.

%% *******************************************************
%% write_cl(Cl) append clases Cl in memory
%% Cl is a list of clauses
%% *******************************************************
write_cl([]).
write_cl([H|T]) :- assert(H),write_cl(T).
```

```
%% ********************************************************
%% search_hyp(L) search synonymous
%% L is a list of synonymou
%% ********************************************************
search_hyp(X,[Word|T],Bd1) :-
hyp(X,Y),Q=hyp,P=..[Q,X,Y],retract(P),db_fetch(Bd1,s(Y,_,Word,_,_),A),!,searc
h_hyp(X,T,Bd1).
search_hyp(X,[],Bd1) :- not(hyp(X,Y)).

%% ********************************************************
%% search_hyp2(L) search synonymous
%% L is a list of synonymous
%% ********************************************************
search_hyp2(Y,[Word|T],Bd1) :-
hyp(X,Y),Q=hyp,P=..[Q,X,Y],retract(P),db_fetch(Bd1,s(X,_,Word,_,_),A),!,searc
h_hyp2(Y,T,Bd1).
search_hyp2(Y,[],Bd1) :- not(hyp(X,Y)).


%% if_subj   See if there is a subject in our list of synonym
if_subj([],nosub).
if_subj([H|T],Hr) :- sub(H,Hr).
if_subj([H|T],Sub) :- not(sub(H,Hr)),if_subj(T,Sub).

if_subj_lim([],nosub,Count).
if_subj_lim([H|T],Hr,Count) :- sub(H,Hr).
if_subj_lim([H|T],Sub,Count) :- Count<10,not(sub(H,Hr)),Countb is
Count+1,if_subj_lim(T,Sub,Countb).
if_subj_lim([H|T],nosub,Count) :- Count>=10.

%% del_sub   Delete all words that have not subject
del_nosub([],[]).
del_nosub([[H1,H2]|T],T1) :- (H2=nosub),del_nosub(T,T1).
del_nosub([[H1,H2]|T],[[H1,H2]|T1]) :- not(H2=nosub),del_nosub(T,T1).

%% quit   pour sortir du pgm
quit([H|_]) :- quit(H).
quit('@@wout') :- setupmode(false).

%%
*************************************************************************
%% r_fi_name(Fi) read the name of file where script will be record
%% Fi is the name of this file
%% ***********************************************************************
r_fi_name(Fi) :- get0(X),rnext(X,Fi).

rnext(X,[])  :- eol(X),!.
rnext(X,[])  :- eof(X),!.
rnext(X,[X|Y]) :- r_fi_name(Y).

%% ***********************************************************************
%% number(X) determine if X est un number or not(Fail)
%% ***********************************************************************
```

```
num(X) :- X >= 48,X =< 57.

%% ************************************************************************
%% clear clear screen
%% ************************************************************************
clear :- print('\f\e[H\e[2J\n').

%% ************************************************************************
%% gen_com_wel(Num,Com) generates a first message for a judge.Message is
%% different if judge arrive for the
%% first time or not
%% Num is number of judge
%% Com is comment to display)
%% ************************************************************************
gen_com_wel(Num,Com) :-
clause((histjud(Num))),intro(again,Com),retract((intro(again,Com))),assertz((
intro(again,Com))).
gen_com_wel(Num,Com) :-
not(clause((histjud(Num)))),intro(new,Com),assert((histjud(Num))),retract((in
tro(new,Com))),assertz((intro(new,Com))).

maj_mode :-
setupmode(true),retract((setupmode(true))),assert((setupmode(false))).
maj_mode :- setupmode(false).

%% ************************************************************************
%% ctrl_input(Input,St) controls if Input is not @@TCRCR or @@nnCRCR
%% Input is the input to control
%% St is the stream in wich scripts are write
%%
************************************************************************
%% int(X) used like boolean ti indicate if prompt is a ?
:- dynamic int/1.
int(false).

ctrl_input([64,64,84],St) :-
                    clear,
                    maj_mode,
                    retract((int(false))),assert((int(true))),
                    %% qcompile(comment),
                    prompt(_,'?'),!.
ctrl_input([64,64,X],St) :- setupmode(true),!.
ctrl_input([64,64,X,Y],St) :- setupmode(true),!.
ctrl_input([64,64,X],St) :-
                        num(X),
                         name(Num,[X]),
                         write(St,'****JUDGE
'),write(St,Num),write(St,'****'),write(St,'\n'),
                    maj_int,
                    qcompile(c),
                    judge(NumJud),
                    retract((judge(NumJud))),
                    name(Numok,[X]),
                    assert((judge(Numok))),
                    clear,
```

```
                    prompt(_,'>'),
                name('Welcome ',Msga),
                name('judge ',Msgb),
                name(Numok,Msgc),
                append(Msga,Msgb,Msgcc),
                append(Msgcc,Msgc,MsgAscok),
                name(Msgok,MsgAscok),
                gen_com_wel(Numok,Com),
                    write_comment(Com,St),!.
ctrl_input([64,64,X,Y],St) :-
                num(X),
                num(Y),
                name(Num,[X,Y]),
                qcompile(c),
                write(St,'****JUDGE
'),write(St,Num),write(St,'****'),write(St,'\n'),
                clear,
                maj_int,
                judge(NumJud),
                retract((judge(NumJud))),
                name(Numok,[X,Y]),
                assert((judge(Numok))),
                prompt(_,'>'),
                name('Welcome ',Msga),
                    name('judge ',Msgb),
                    name(Numok,Msgc),
                    append(Msga,Msgb,Msgcc),
                    append(Msgcc,Msgc,MsgAscok),
                    name(Msgok,MsgAscok),
                gen_com_wel(Numok,Com),
                    write_comment(Com,St),!.
ctrl_input(_,St).

%% ******************************
%% to check the name of file
%% ******************************
ctrl_name([],[116,101,115,116]).
ctrl_name([H|T],[H|T]).

%% ************************************************************************
%% pr_file(Sent,Fi) write a sentence in file Fi if Sent is note a special
code
%% (@@...)
%% Sent is a list
%% ************************************************************************
pr_file([],Fi).
pr_file([H|T],Fi) :-   name(H,[64,64,84]),!.
pr_file([H|T],Fi) :-   name(H,[64,64,X]),num(X),!.
pr_file([H|T],Fi) :-   name(H,[64,64,X,Y]),num(X),num(Y),!.
pr_file([H|T],Fi) :-   name(H,[64,64,120]),!.
pr_file([H|T],Fi) :-   name(H,[64,64,110,110]),!.
pr_file([H|T],Fi) :-   nl,write(Fi,H),write(Fi,' '),pr_file(T,Fi).

%% ************************************************************************
%% verif_input(Input) verify user cannot type other things that @@nn or @@Xif
```

```
%% prompt is ?
%% this function uses int(X) X=true or false to determinate if prompt is ?
%% ************************************************************************
verif_input(Input) :- int(true),(Input=[64,64,X,Y]),num(X),num(Y).
verif_input(Input) :- int(true),(Input=[64,64,X]),num(X).
verif_input(Input) :- int(true),(Input=[64,64,88]).
verif_input(Input) :- int(false).

%% ************************************************************************
%% maj_int maintain int(X) in a coherent state (int(true) if we have the
prompt
%% ?
%% ************************************************************************
maj_int :- int(true),retract((int(true))),assert((int(false))).
maj_int :- int(false).

%% ************************************************************************
%% r_inp_ver(Input) read Input and verify if it's valid
%% Input is string read
%% ************************************************************************
r_inp_ver(Input,St) :-
      repeat,
      d_input(Input,[],St),
            verif_input(Input),
      !.

%% ************************************************************************
%% ver_cht_jud(Input,Inputb) verify si Input is a cht of judge (@@n or
%% @@nn).If yes,then Input b is the special code '@@num',if no Input b is
%% Input
%% ************************************************************************
ver_cht_jud([64,64,X],[64,64,110,110])    :- num(X),!.
ver_cht_jud([64,64,110,110],[110,110])    :- !.
ver_cht_jud([64,64,X,Y],[64,64,110,110]) :- num(X),num(Y),!.
ver_cht_jud([64,64,119,111,117,116],[119,111,117,116]) :- !.
ver_cht_jud([64,64,X],[64,64,119,111,117,116]) :- X=88,!.
ver_cht_jud(In,Inb)               :- (Inb=In).

%% ************************************************************************
%% extr_time(H,M,S) extract the time
%% ************************************************************************
:-dynamic datime/6.
extr_time(H,M,S) :- datime(T),assert((T)),datime(Y,Mo,D,H,M,S),retract((T)).

%%************************************************************************
%% cr_l_subj(K,L)  takes our keywords and finds subjects and no subjects.
%%************************************************************************
cr_l_subj([],[],[]).
cr_l_subj([H|T],[Hr|Y],Z) :- sub(H,Hr),cr_l_subj(T,Y,Z).
cr_l_subj([H|T],Sub,[H|Z]) :-
not(sub(H,Hr)),plural_sub(H,Sg),Sg=notfind,cr_l_subj(T,Sub,Z).
cr_l_subj([H|T],[Sgr|Y],Z) :-
not(sub(H,Hr)),plural_sub(H,Sg),not(Sg=notfind),sub(Sg,Sgr),cr_l_subj(T,Y,Z).
```

```
cr_l_subj([H|T],Sub,[H|Z]) :-
not(sub(H,Hr)),plural_sub(H,Sg),not(Sg=notfind),not(sub(Sg,Sgr)),cr_l_subj(T,
Sub,Z).

%%*************************************************************************
%% plural_sub(K,L)  takes a list of words and gives words plus each singular.
%%*************************************************************************
plur_sing([],[]).
plur_sing([H|T],[H|Y]) :- plural_sub(H,S),S=notfind,plur_sing(T,Y).
plur_sing([H|T],[H,S|Y]) :- plural_sub(H,S),not(S=notfind),plur_sing(T,Y).

%%*********************************************************
%% plural_sub(K,L)  takes word and gives singular or notfind.
%%*********************************************************
plural_sub(Pl,Sg) :-
name(Pl,Lp),last_elm(Lp,Elm),Elm=115,dele_last(Lp,Ls),name(Sg,Ls).
plural_sub(Pl,Sg) :- name(Pl,Lp),last_elm(Lp,Elm),not(Elm=115),Sg=notfind.

%%***********************************************
%% del_last(K,L)  deletes last element of a list.
%%***********************************************
dele_last([H],[]).
dele_last([H|T],[H|Y]) :- not(T=[]),dele_last(T,Y).

%%*********************************************
%% last_elm(K,L)  gives last element of a list.
%%*********************************************
last_elm([H],H).
last_elm([H|T],Elm) :- not(T=[]),last_elm(T,Elm).

%% ************************************************************************
%% wr_(Com,St) write the commentaries into the stream St
%%
*************************************************************************
write_fi([],Beg,_).
write_fi([H|T],Beg,St) :-
(H=10),name(Com,Beg),wr_com_fi(St,Com),(X=[]),write_fi(T,X,St).
write_fi([H|T],Beg,St) :- not(H=10),append(Beg,[H],Beg2),write_fi(T,Beg2,St).

%% ************************************************************************
%% verif_num(Num,St) verify Num is a number with 2 cifers and print a 0
before
%% in the stream if no
%% Num is a number
%% St is the stream
%% ************************************************************************
verif_num(Num,St) :- name(Num,Numasc),length(Numasc,L),L=1,write(St,0).
verif_num(Num,St) :- name(Num,Numasc),length(Numasc,L),L=2.

%% ************************************************************************
%% wr_com_fi(St,Com) write the commentary into the stream St
%% ************************************************************************
wr_com_fi(St,'').
wr_com_fi(St,'@@t') :- write(St,'\n'),!.
```

```
wr_com_fi(St,'@@x') :- write(St,'\n'),!.
wr_com_fi(St,X) :-
not(X=''),write(St,'PROGRAM'),extr_time(H,M,S),write(St,'['),verif_num(H,St),
write(St,H),write(St,':'),verif_num(M,St),write(St,M),write(St,':'),verif_num
(S,St),write(St,S),write(St,']'),write(St,X),write(St,'\n').

%%*********************************************************************
%% Fermeture des bases de donnees
%%*********************************************************************
close_db(Bd1,Bd2,Bd3,Bd4) :-
db_close(Bd1),db_close(Bd2),db_close(Bd3),db_close(Bd4).

%%*********************************************************************
%%  On extrait une sous-liste d'une liste
%%  C'est utile pour afficher le comm. d'attente si on ne
%%  trouve pa de syn apres 3 essais de mot cles
%%*********************************************************************
subllist([],[],_).
subllist([H|T],[],X) :- X=5.
subllist([H|T],[H|Y],X) :- not(X=5),Z is X + 1,subllist(T,Y,Z).

%% ************************************************************************
%% To put a new state for randoms numbers each time that we launch program
%% ************************************************************************
:- dynamic datime/6.
state_random(X) :- assert((X)),datime(Y,M,D,H,Mi,S),Arg1=M,Arg2 is D + H,Arg3
is Mi + S,setrand(rand(Arg1,Arg2,Arg3)),retract((X)).

:- open('/dev/null',write,S), assert(null(S)).
quiet(X):- null(S), prolog_flag(user_error,X,S).
noisy(X):- prolog_flag(user_error,_,X).
qconsult(L):- quiet(X), consult(L), noisy(X).
qcompile(L):- quiet(X), compile(L), noisy(X).
qload(L):- quiet(X), load(L), noisy(X).

%% ************************************************************************
%% To print year,month,day,hour,minute,second in the beginning of the file
%% ************************************************************************
print_time(St,X) :- assert((X)),datime(Y,M,D,H,Mi,S),print(St,'Start at:
[',print(St,Y),print(St,'/'),verif_num(M,St),print(St,M),print(St,'/'),verif
_num(D,St),print(St,D),print(St,'
'),verif_num(H,St),print(St,H),print(St,':'),verif_num(Mi,St),print(St,Mi),pr
int(St,':'),verif_num(S,St),print(St,S),print(St,']\n'),retract((X)).

%% ******************************
%% To delete dubbles inside a list
%% del_dubble(X,Y).
%% X is the list where we take off dubble
%% Y is the list without dubble
%% ******************************
del_dubble([],[]).
del_dubble([H|T],Z) :- scan_dubble(H,T,Find),Find=1,del_dubble(T,Z).
del_dubble([H|T],[H|Z]) :- scan_dubble(H,T,Find),Find=0,del_dubble(T,Z).
```

```
%% ************************************************************
%% To check if the element exist again in the rest of the list
%% scan_dubble(X,Y,Z).
%% X is the word to find in the list
%% Y is the list where we search
%% Z is like a boolean 1 if we found 0 if we didn't find
%% ************************************************************
scan_dubble(_,[],0).
scan_dubble(H,[H|T],1).
scan_dubble(X,[H|T],Find) :- not(X=H),scan_dubble(X,T,Find).

%% ****************** SEARCH SUB ****************
%% *********************************************
%% To search the comment that matches better but only for subject
%% research_comment_sub(X,Y,Z,Par).
%% X is the first word of the sentence to see if it's an interrogation ...
%% Y is the list of subjects that we found in the sentence
%% Z is the list of the clause of comment that matches the subjects
%% Par is the parameter if we search for gener_com_quest or gener_com_aff
%%     quest -> gener_com_quest
%%     aff -> gener_com_aff
%% ************************************************************************
:- dynamic clause_sub/2.
:- dynamic best_clause_sub/2.

research_comment_sub(FirstWord,[],[],_).
research_comment_sub(FirstWord,[H|T],L,Par) :-
Lsub=[H|T],search_clause_fw(FirstWord,Lfw,Par),search_clause_sub(Lsub,Lfw,Par
),assert((best_clause_sub(xx,0))),keep_best_clause_sub,retractall((clause_sub
(A,B))),cr_l_clause_sub(L).

%% ************************************************************************
%% To search clauses with this first word
%% search_clause_fw(X,Y,Par).
%% X is the first word of the sentence to see if it's an interrogation ...
%% Y is the list of sentences matching with this word
%% Par is the parameter if we search for gener_com_quest, gener_quest or
%% gener_com_aff
%% ************************************************************************
search_clause_fw(FirstWord,Lst,Par) :-
Par=quest,call_sub_quest(FirstWord,Lst).
search_clause_fw(FirstWord,Lst,Par) :- Par=aff,call_sub_aff(FirstWord,Lst).

call_sub_quest(FirstWord,Lst) :-
findall(Num,gener_com_quest(Num,FirstWord,L,_,_),Lst).
call_sub_quest(FirstWord,[]) :-
not(findall(Num,gener_com_quest(Num,FirstWord,L,_,_),Lst)).

call_sub_aff(_,Lst) :- findall(Num,gener_com_aff(Num,L,_,_),Lst).
call_sub_aff(_,[]) :- not(findall(Num,gener_com_aff(Num,L,_,_),Lst)).

%% ************************************************************************
%% To search clauses with subjects and we put these clause in MEMORY
%% search_clause_sub(X,Y).
%% X is the list of subjects that we found in the sentence
```

```
%% Y is the list of the clause that matched the first word of the sentence
%% ************************************************************************
search_clause_sub([H|T],[],_).
search_clause_sub([],[],_).
search_clause_sub([],[H|T],_).
search_clause_sub(Lsub,[Num|T],quest) :-
not(Lsub=[]),gener_com_quest(Num,_,L,_,_),comp(Lsub,L,0,Nb),not(Nb=0),assert(
(clause_sub(Num,Nb))),search_clause_sub(Lsub,T,quest).
search_clause_sub(Lsub,[Num|T],quest) :-
not(Lsub=[]),gener_com_quest(Num,_,L,_,_),comp(Lsub,L,0,Nb),Nb=0,search_claus
e_sub(Lsub,T,quest).
search_clause_sub(Lsub,[Num|T],aff) :-
not(Lsub=[]),gener_com_aff(Num,L,_,_),comp(Lsub,L,0,Nb),not(Nb=0),assert((cla
use_sub(Num,Nb))),search_clause_sub(Lsub,T,aff).
search_clause_sub(Lsub,[Num|T],aff) :-
not(Lsub=[]),gener_com_aff(Num,L,_,_),comp(Lsub,L,0,Nb),Nb=0,search_clause_su
b(Lsub,T,aff).

%% *******************************************************
%% To take the best clause with the most subjects matching
%% keep_best_clause_sub.
%% *******************************************************
keep_best_clause_sub :- not(clause_sub(X,Y)).
keep_best_clause_sub :-
clause_sub(X,Y),best_clause_sub(A,B),Y>B,retractall((best_clause_sub(_,B))),a
ssert((best_clause_sub(X,Y))),retract((clause_sub(X,Y))),keep_best_clause_sub
.
keep_best_clause_sub :-
clause_sub(X,Y),best_clause_sub(A,B),Y<B,retract((clause_sub(X,Y))),keep_best
_clause_sub.
keep_best_clause_sub :-
clause_sub(X,Y),best_clause_sub(A,B),Y=B,assert((best_clause_sub(X,Y))),retra
ct((clause_sub(X,Y))),keep_best_clause_sub.

%% ************************************************************************
%% To compare two lists and give the number of same elements
%% comp(W,X,Y,Z).
%% W is the list of subjects that we found in the sentence
%% X is the list that we found in the clause of comment
%% Y is an element that serves of counter
%% Z is the number of element of the list W that we find in the list X
%% ************************************************************************
comp([],L,Temp,Nb) :- Nb=Temp.
comp([H|T],L,Temp,Nb) :- memb(H,L,N),Z is Temp+N,comp(T,L,Z,Nb).

%% ********************************************************************
%% To create a list with clause subjects
%% cr_l_clause_sub(X).
%% X is the list of (Number of clause,Number of subjects matching for this
%% clause) that we create
%% ********************************************************************
cr_l_clause_sub([]) :-
best_clause_sub(X,Y),X=xx,Y=0,retract((best_clause_sub(X,Y))).
cr_l_clause_sub([]) :- not(best_clause_sub(X,Y)).
```

```
cr_l_clause_sub([X|T]) :-
best_clause_sub(X,Y),not(X=xx),not(Y=0),retract((best_clause_sub(X,Y))),cr_l_
clause_sub(T).


%% ***************************************SEARCH NOSUB **********************
%% To search the comment that matches better but for no subject
%% research_comment_nosub(X,Y,Z,Par).
%% X is the list of no subjects
%% Y is the list of clause that matched for subjects
%% Par is the parameter if we search for gener_com_quest, gener_quest or
%% gener_com_aff
%% ************************************************************************
:- dynamic clause_nosub/2.
:- dynamic best_clause_nosub/2.
research_comment_nosub([],L,_).
research_comment_nosub([H|T],Lcl,Par) :-
search_clause_nosub([H|T],Lcl,Par),assert((best_clause_nosub(xx,-
1))),keep_best_clause_nosub,retractall((clause_nosub(A,B))).

%% ************************************************************************
%% To search clauses with no subjects and we put these clause in MEMORY
%% search_clause_nosub(X,Y,Par).
%% X is the list of no subjects that we found in the sentence
%% Y is the list of the clause that matched the subjects of the sentence
%% Par is the parameter if we search for gener_com_quest, gener_quest or
%% gener_com_aff
%% If we don't find clauses that matches with no subjects, we must have a
%% comment and it's for that
%% we keep clauses with a comp that gives 0.
%% ************************************************************************
search_clause_nosub([H|T],[],_).
search_clause_nosub([],[],_).
search_clause_nosub([],[H|T],_).
search_clause_nosub(Lnosub,[Num|T],quest) :-
not(Lnosub=[]),gener_com_quest(Num,_,_,L,_),comp(Lnosub,L,0,Nb),assert((claus
e_nosub(Num,Nb))),search_clause_nosub(Lnosub,T,quest).
search_clause_nosub(Lnosub,[Num|T],aff) :-
not(Lnosub=[]),gener_com_aff(Num,_,L,_),comp(Lnosub,L,0,Nb),assert((clause_no
sub(Num,Nb))),search_clause_nosub(Lnosub,T,aff).

%% ******************************************************
%% To take the best clause with the most subjects matching
%% keep_best_clause_nosub.
%% ******************************************************
keep_best_clause_nosub :- not(clause_nosub(X,Y)).
keep_best_clause_nosub :-
clause_nosub(X,Y),best_clause_nosub(A,B),Y>B,retractall((best_clause_nosub(_,
B))),assert((best_clause_nosub(X,Y))),retract((clause_nosub(X,Y))),keep_best_
clause_nosub.
keep_best_clause_nosub :-
clause_nosub(X,Y),best_clause_nosub(A,B),Y<B,retract((clause_nosub(X,Y))),kee
p_best_clause_nosub.
```

```
keep_best_clause_nosub :-
clause_nosub(X,Y),best_clause_nosub(A,B),Y=B,assert((best_clause_nosub(X,Y)))
,retract((clause_nosub(X,Y))),keep_best_clause_nosub.

%% ************************************************************
%% To take the best comment that matches the best
%% take_best_comment(X).
%% X is the comment that matches the best with the input sentence
%% ************************************************************
take_best_comment(C,_) :- best_clause_nosub(X,Y),X=xx,Y=0. %% normally never
happens
take_best_comment(C,quest) :-
best_clause_nosub(X,Y),not(X=xx),gener_com_quest(X,Fw,A,B,C),retract((gener_c
om_quest(X,Fw,A,B,C))),assertz((gener_com_quest(X,Fw,A,B,C))),retractall((bes
t_clause_nosub(I,J))),!.
take_best_comment(C,aff) :-
best_clause_nosub(X,Y),not(X=xx),gener_com_aff(X,A,B,C),retract((gener_com_af
f(X,A,B,C))),assertz((gener_com_aff(X,A,B,C))),retractall((best_clause_nosub(
I,J))),!.

%%*****************************************************************************
***************
%%% Module principal %%%%%%%%%%
%%*****************************************************************************
***************
:- dynamic library/1.
:- dynamic judge/1.
:- dynamic histjud/1.
:- dynamic setupmode/1.
el :-
        retractall((int(X))),assert((int(false))),
        retractall((judge(X))),
        retractall((histjud(X))),
        retractall((setupmode(X))),
        retractall((best_clause_sub(A,B))),
        retractall((best_clause_nosub(E,F))),
        retractall((clause_sub(C,D))),
        retractall((clause_nosub(G,H))),
        use_module(library(random)),
        use_module(library(system)),
        use_module(library(timeout)),
        qcompile(comment),
        qcompile(keyprox),
        qcompile(wnal),
        qcompile(introeli),
        tell(user_error),
        ouv_bd(Bd1,Bd2,Bd3,Bd4),
        prompt(_, '+'),
        print('Please type a file name : '),
        r_fi_name(Fi),
        ctrl_name(Fi,Nfi),
        name(Nfifi,Nfi),
        open(Nfifi,write,St), %% opening of log file
        print(St,'(c)1998 Cambridge Center For Behavorioral Studies all
rights reserved\n'),
```

```
        print(St,'[ELISABETH]  - [REALIZED BY V.BASTIN AND D.CORDIER]\n'),
        datime(X),
        state_random(X),
        print_time(St,X),
        print(St,'****JUDGE00*****\n'),
        assert((judge(00))),
        assert((histjud(00))),
        assert((setupmode(true))),
        clear,
        intro(new,Com),
        write_comment(Com,St),
        retract((intro(new,Com))),
        assertz((intro(new,Com))),
        repeat,
         r_inp_ver(Input,St),
         ctrl_input(Input,St),
         ver_cht_jud(Input,Inputv),
         extract_input(Inputv,I),
         low(Inputv,InputB),
         clean_string(InputB,InputC),
         extract_input(InputC,InputD),
         simpa(InputD,InputDDD),
         simplify(InputDDD,InputE),
         find_key(InputE,KeyWords),
            %% we search some keywords in the sentence
          plur_sing(KeyWords,KeyWords2),
          cr_l_subj(KeyWords2,Ksubj,Nosubj),
            %% We create a list of subjects and a list of no subjects
          find_syn_hyp(Nosubj,InputE,Lstay,Lsub,Bd1,Bd2),

            %% We search for synonyms and hypernyms but for the 3 first
            %% keywords or if it's a predefined sentence
         append(Ksubj,Lsub,Llsub),  %% We concatenate subjects that we found
          del_dubble(Llsub,Lllsub), %% We delete all dubbles
          length(Lllsub,Long),
            %% We take the number of subjects that we found
          search_syn(Lsub,Lstay,Long,Lllsub,Res,St,Bd1,Bd2),
            %% If the number of subjects is one, we continue
            %% reseearch of synonyms and hypernyms on others keywords
            %% before we check Lsub because the sentence can be predef
          append(Lllsub,Res,Lisub), %% We concatenate subjects
          search_sub_verb(KeyWords2,Lverb),
            %% We search for subjects but for verbs
          append(Lisub,Lverb,Listsub), %% We concatenate subjects
          del_dubble(Listsub,Lsubject), %% We delete all dubbles

make_comment(Lsubject,Nosubj,InputE,KeyWords,Bd1,Bd2,Bd3,Bd4,Comment,St),
          P=..[red_quest,vide,dorspas,''],assertz((P)),
            %% to be sure that there is enough red_quest,
          quit(InputE),
      !,
      close(St),
      close_db(Bd1,Bd2,Bd3,Bd4).

save_el(X) :- muse_flag(num_workers,N,1), save(X),
muse_flag(num_workers,_,N), el.
```

2. Commentaires pré-définis

```
%% Sub - differents subject of speaking
sub(animal,animal).
sub(work,work).
sub(holiday,holiday).
sub(belgium,belgium).
sub(belgian,belgium).
sub(liege,belgium).
sub(bruxelles,belgium).
sub(namur,belgium).
sub(charleroi,belgium).
sub(gand,belgium).
sub(bruges,belgium).
sub(anvers,belgium).
sub(school,school).
sub(university,school).
sub(student,school).
sub(australia,australia).
sub(australian,australia).
sub(adelaide,australia).
sub(sydney,australia).
sub(canberra,australia).
sub(perth,australia).
sub(darwin,australia).
sub(brisbane,australia).
sub(cairn,australia).
sub(melbourne,australia).
sub(family,family).
sub(parents,family).
sub(husband,family).
sub(wife,family).
sub(mother,family).
sub(father,family).
sub(brother,family).
sub(sister,family).
sub(godmother,family).
sub(godfather,family).
sub(grandparents,family).
sub(grandmother,family).
sub(grandfather,family).
sub(aunt,family).
sub(uncle,family).
sub(cousin,family).
sub(families,family).
sub(religion,religion).
sub(mosque,religion).
sub(church,religion).
sub(churches,religion).
sub(christian,religion).
sub(apostolic,religion).
```

```
sub(baptist,religion).
sub(catholic,religion).
sub(anglican,religion).
sub(priest,religion).
sub(pastor,religion).
sub(baptism,religion).
sub(hobby,hobby).
sub(hobbies,hobby).
sub(interest,hobby).
sub(language,lang).
sub(bonjour,lang).
sub(salut,lang).
sub(aurevoir,lang).
sub(merci,lang).
sub(morgen,lang).
sub(dag,lang).
sub(dank,lang).
sub(name,name).
sub(firstname,firstname).
sub(birthday,birthday).
sub(david,noun).
sub(sandra,noun).
sub(claire,noun).
sub(stephanie,noun).
sub(walter,noun).
sub(mark,noun).
sub(dominique,noun).
sub(christer,noun).
sub(harold,noun).
sub(diana,noun).
sub(ingrid,noun).
sub(michael,noun).
sub(thorsten,noun).
sub(claudia,noun).
sub(stephan,noun).
sub(stephen,noun).
sub(greg,noun).
sub(ilyas,noun).
sub(hamish,noun).
sub(herve,noun).
sub(erika,noun).
sub(joachim,noun).
sub(gerard,noun).
sub(james,noun).
sub(patrick,noun).
sub(andre,noun).
sub(kevin,noun).
sub(christopher,noun).
sub(richard,noun).
sub(ingo,noun).
sub(rene,noun).
sub(isabelle,noun).
```

```
sub(ian,noun).
sub(antal,noun).
sub(ton,noun).
sub(edmund,noun).
sub(fred,noun).
sub(paul,noun).
sub(hugh,noun).
sub(denis,noun).
sub(veronique,noun).
sub(vero,noun).
sub(catherine,noun).
sub(sue,noun).
sub(jim,noun).
sub(charles,noun).
sub(john,noun).
sub(carl,noun).
sub(peter,noun).
sub(chris,noun).
sub(susan,noun).
sub(sam,noun).
sub(robert,noun).
sub(simon,noun).
sub(graham,noun).
sub(janet,noun).
sub(tiffany,noun).
sub(kelly,noun).
sub(allan,noun).
sub(alan,noun).
sub(jason,noun).
sub(luke,noun).
sub(jennifer,noun).
sub(tim,noun).
sub(bruce,noun).
sub(bradley,noun).
sub(brad,noun).
sub(caroline,noun).
sub(monica,noun).
sub(jack,noun).
sub(elisabeth,eli).
sub(elizabeth,eli).
sub(elisa,eli).
sub(eliza,eli).
sub(car,car).
sub(auto,car).
sub(holden,car).
sub(ford,car).
sub(jeep,car).
sub('4x4',car).
sub(toyota,car).
sub(mitsubishi,car).
sub(nissan,car).
sub(bicycle,car).
```

```
sub(moto,car).
sub(motorbyke,car).
sub(vehicle,car).
sub(transport,transport).
sub(plane,transport).
sub(bus,transport).
sub(truck,transport).
sub(train,transport).
sub(coach,transport).
sub(abortion,debat).
sub(aids,debat).
sub(racism,debat).
sub(spatial,debat).
sub(murderer,debat).
sub(unemployment,debat).
sub(pollution,debat).
sub(nuclear,debat).
sub(greenhouse,debat).
sub(crash,debat).
sub(accident,debat).
sub(disaster,debat).
sub(shipwreck,debat).
sub(murder,debat).
sub(rape,debat).
sub(kidnapping,debat).
sub(science,science).
sub(physics,science).
sub(biology,science).
sub(chemistry,science).
sub(mathematics,science).
sub(spatial,science).
sub(economy,science).
sub(game,game).
sub(chess,game).
sub(lotto,game).
sub(monopoly,game).
sub(boggle,game).
sub(casino,game).
sub(bet,game).
sub(lottery,game).
sub(lotto,game).
sub(titanic,titanic).
sub(culture,culture).
sub(theatre,culture).
sub(cinema,culture).
sub(movie,culture).
sub(song,culture).
sub(guitar,culture).
sub(beatles,culture).
sub(violin,culture).
sub(saxophone,culture).
sub(piano,culture).
```

```
sub(trumpet,culture).
sub(opera,culture).
sub(singer,culture).
sub(television,culture).
sub(radio,culture).
sub(video,culture).
sub(concert,culture).
sub(cd,culture).
sub(museum,culture).
sub(painting,culture).
sub(art,culture).
sub(sculptor,culture).
sub(sculpture,culture).
sub(actor,culture).
sub(actress,culture).
sub(film,culture).
sub(book,culture).
sub(writer,culture).
sub(painter,culture).
sub(drawing,culture).
sub(artist,culture).
sub(drawer,culture).
sub(profession,profession).
sub(vet,profession).
sub(doctor,profession).
sub(teacher,profession).
sub(scientist,profession).
sub(surgeon,profession).
sub(mechanic,profession).
sub(servant,profession).
sub(lawyer,profession).
sub(professor,profession).
sub(secretary,profession).
sub(nurse,profession).
sub(hairdresser,profession).
sub(beach,beach).
sub(bondai,beach).
sub(manly,beach).
sub(glenelg,beach).
sub(brighton,beach).
sub(beaches,beach).
sub(sea,beach).
sub(sunburn,beach).
sub(sun,beach).
sub(sand,beach).
sub(boat,beach).
sub(wave,beach).
sub(sunrise,beach).
sub(tide,beach).
sub(surf,beach).
sub(drinks,drinks).
sub(soda,drinks).
```

```
sub(coca,drinks).
sub(lemonade,drinks).
sub(coffee,drinks).
sub(tea,drinks).
sub(water,drinks).
sub(ginger,drinks).
sub(alcohol,alcohol).
sub(beer,alcohol).
sub(wine,alcohol).
sub(gin,alcohol).
sub(vodka,alcohol).
sub(whisky,alcohol).
sub(sport,sport).
sub(tennis,sport).
sub(squash,sport).
sub(football,sport).
sub(soccer,sport).
sub(skating,sport).
sub(ski,sport).
sub(cricket,sport).
sub(bowling,sport).
sub(volley,sport).
sub(polo,sport).
sub(food,food).
sub(kiwi,food).
sub(banana,food).
sub(appel,food).
sub(orange,food).
sub(manga,food).
sub(pasta,food).
sub(raspberry,food).
sub(strawberry,food).
sub(mussel,food).
sub(restaurant,food).
sub(beef,food).
sub(pork,food).
sub(lamb,food).
sub(chicken,food).
sub(potato,food).
sub(vegetable,food).
sub(chip,food).
sub(carrot,food).
sub(capsikon,food).
sub(brocoli,food).
sub(brocolli,food).
sub(corn,food).
sub(cream,food).
sub(lettuce,food).
sub(bread,food).
sub(ham,food).
sub(bacon,food).
sub(cheese,food).
```

```
sub(feast,feast).
sub(christmas,feast).
sub(easter,feast).
sub(marriage,feast).
sub(wedding,feast).
sub(eve,feast).
sub(new-york,ville).
sub(usa,ville).
sub(europe,ville).
sub(china,ville).
sub(france,ville).
sub(england,ville).
sub(italy,ville).
sub(germany,ville).
sub(africa,ville).
sub(paris,ville).
sub(london,ville).
sub(washington,ville).
sub(island,ville).
sub(moscow,ville).
sub(man,pers).
sub(woman,pers).
sub(boy,pers).
sub(girl,pers).
sub(male,pers).
sub(female,pers).
sub(tired,tired).
sub(angry,silly).
sub(silly,silly).
sub(stupid,silly).
sub(idiot,silly).
sub(computer,computer).
sub(competition,competition).
sub(loebner,competition).
sub(money,money).
sub(child,child).
sub(children,child).
sub(time,time).
sub(hour,time).
sub(minute,time).
sub(second,time).
sub(month,date).
sub(day,date).
sub(year,date).
sub(monday,date).
sub(tuesday,date).
sub(wednesday,date).
sub(thursday,date).
sub(friday,date).
sub(saturday,date).
sub(sunday,date).
sub(january,date).
```

```
sub(february,date).
sub(march,date).
sub(april,date).
sub(may,date).
sub(june,date).
sub(july,date).
sub(augustus,date).
sub(september,date).
sub(october,date).
sub(november,date).
sub(december,date).
sub(phone,phone).
sub(body,body).
sub(hair,body).
sub(eye,body).
sub(nose,body).
sub(tall,body).
sub(weather,weather).
sub(cold,weather).
sub(warm,weather).
sub(hot,weather).
sub(winter,weather).
sub(spring,weather).
sub(summer,weather).
sub(autumn,weather).
sub(snow,weather).
sub(rain,weather).
sub(friend,friend).
sub(mate,friend).
sub(ami,friend).
sub(pal,friend).
sub(powerhouse,powerhouse).
sub(strasburg,powerhouse).
sub(locomotive,powerhouse).
sub(startrek,powerhouse).
sub(trek,powerhouse).
sub(cyberzone,powerhouse).
sub(circus,powerhouse).
sub(dress,powerhouse).
sub(herald,powerhouse).
sub(ngaramang,powerhouse).
sub(bayumi,powerhouse).
sub_verb(sing,culture).
sub_verb(singing,culture).
sub_verb(draw,culture).
sub_verb(talk,talk).
sub_verb(talking,talk).
sub_verb(talked,talk).
sub_verb(speak,talk).
sub_verb(speaking,talk).
sub_verb(spoke,talk).
sub_verb(say,talk).
```

```
sub_verb(saying,talk).
sub_verb(said,talk).
sub_verb(tell,talk).
sub_verb(telling,talk).
sub_verb(told,talk).
sub_verb(run,walk).
sub_verb(running,walk).
sub_verb(ran,walk).
sub_verb(walk,walk).
sub_verb(walking,walk).
sub_verb(walked,walk).
sub_verb(see,see).
sub_verb(seeing,see).
sub_verb(saw,see).
sub_verb(look,see).
sub_verb(looking,see).
sub_verb(looked,see).
sub_verb(hear,hear).
sub_verb(hearing,hear).
sub_verb(heard,hear).
sub_verb(listen,hear).
sub_verb(listening,hear).
sub_verb(listened,hear).
sub_verb(eat,eat).
sub_verb(eating,eat).
sub_verb(ate,eat).
sub_verb(drink,eat).
sub_verb(drinking,eat).
sub_verb(drank,eat).
sub_verb(taste,eat).
sub_verb(tasting,eat).
sub_verb(tasted,eat).
sub_verb(ride,ride).
sub_verb(riding,ride).
sub_verb(born,born).
sub_verb(travel,ville).
sub_verb(sleep,sleep).
sub_verb(sleeping,sleep).
sub_verb(slept,sleep).
sub_verb(dream,sleep).
sub_verb(dreaming,sleep).
sub_verb(dreamed,sleep).
sub_verb(swim,beach).
sub_verb(swimming,beach).
sub_verb(swam,beach).
sub_verb(cheat,cheat).
sub_verb(cheating,cheat).

%% ************************************************************************
%% our predefined sentences
%% ************************************************************************
```

```
predef([thank,i|Suite],'No problem.') .
predef([ta|Suite],'It\'s fine.') .
predef([how,am,i|Suite],'Very fine, thanks.') .
predef([how,old|Suite],'I\'m 23 years old.') .
predef([describe,myself|Suite],'I prefer to not speak about my body.') .
predef([what,old|Suite],'I\'m 23 years old.') .
predef([you,think|Suite],'I wait your response.') .
predef([you,dont,know|Suite],'Why?') .
predef([do,i,know|Suite],'Doesn\'t matter in this competition.') .
predef([perhaps|Suite],'Yes or no?') .
predef([thank,you|Suite],'No problem.') .
predef([it,is,true|Suite],'We have the same opinion.') .
predef([that,is,true|Suite],'Ok, you agree with me.') .
predef([maybe|Suite],'Yes or no?') .
predef([hi|Suite],'Today I\'m very happy to speak with you. Have you
animals?') .
predef([hello|Suite],'Hello, it\'s a nice day, isn\'t it?') .
predef([good,morning|Suite],'I\'m very happy to speak with you. Do you like
cinema') .
predef([morning|Suite],'Ah, I\'m very fine today and I\'m happy to speak with
you.') .
predef([g,day|Suite],'Hello, it\'s a nice day to speak but I will prefer to
speak with you on the beach') .
predef([g,dday|Suite],'Hello, it\'s a nice day to speak but I will prefer to
speak with you on the beach') .
predef([and,i|Suite],'Doesn\'t matter.') .
predef([yes],'Ok.') .
predef([ok|Suite],'You speak a lot today.') .
predef([no],'No problem.') .
predef([hello,how,am,i|Suite],'Very fine, thanks.') .
predef([quit|Suite],'Bye, nice to meet you.') .
predef([bye|Suite],'Bye, nice to meet you.') .
predef([see,i|Suite],'Bye, I hope to see you again.') .
predef([good,night|Suite],'You are going to sleep now. It\'s not a bit
early.') .
predef([good,evening|Suite],'It\'s so late, I must leave soon.') .
predef([good,afternoon|Suite],'Good afternoon to you too. Did you have a good
day?') .
predef([goodbye|Suite],'Bye,nice to meet you.') .
predef([ciao|Suite],'Aurevoir, if you speak Italian I speak French.') .
predef([stop|Suite],'Bye,nice to meet you.') .
predef(['@@wout'|Suite],'Bye,nice to meet you.') .
predef([q|Suite],'Bye,nice to meet you.') .
predef([fine|Suite],'Good,I\'m happy of that.') .
predef([very,fine|Suite],'Good,I\'m happy of that.') .
predef([bad|Suite],'Ho, I\'m really sorry of that.') .
predef([very,bad|Suite],'Ho, I\'m really sorry of that.') .
predef([nice,to,meet,i|Suite],'It\'s the same for me to meet you.') .
predef([nice,to,see,i|Suite],'I can\'t see you but I\'m very happy to speak
with you.') .
predef([to,see,i|Suite],'I\'m very happy to speak with you.') .
```

```
predef([to,meet,i|Suite],'It\'s very nice t meet other people, isn\'t it?')
.

:- dynamic red_quest/3.
%% question to redirect the discussion
red_quest(vide,dorspas,'').
red_quest(vide,dorspas,'').
red_quest(work,_,'What\'s your work?').
red_quest(vide,dorspas,'').
red_quest(vide,dorspas,'Have you a lot of friends?').
red_quest(vide,dorspas,'').
red_quest(holiday,_,'Where did you go on holidays last year?').
red_quest(family,_,'Have you a big family?').
red_quest(vide,dorspas,'').
red_quest(holiday,_,'Can you go on holiday this year?').
red_quest(vide,dorspas,'').
red_quest(vide,dormir,'Sorry, I\'m going to toilets').
red_quest(vide,dorspas,'').
red_quest(religion,_,'What\'s your religion?').
red_quest(beach,_,'Do you like the beach?').
red_quest(vide,dorspas,'').
red_quest(game,_,'Had you some prefernces for games?').
red_quest(vide,dorspas,'').
red_quest(sport,_,'Do you like sports?').
red_quest(vide,dorspas,'And you?').
red_quest(vide,dorspas,'').
red_quest(school,_,'Do you know Namur?It\'s the town where I go to the
school.').
red_quest(vide,dorspas,'').
red_quest(vide,dorspas,'').
red_quest(culture,_,'Did you see Titanic?').
red_quest(vide,dorspas,'').
red_quest(hobby,_,'What kind of hobbies do you like?').
red_quest(computer,_,'Have you a computer?').
red_quest(vide,dorspas,'').
red_quest(belgium,_,'Do you know Belgium?').
red_quest(vide,dorspas,'').
red_quest(lang,_,'Do you know French?').
red_quest(ville,_,'Have you ever been in Europe?').
red_quest(vide,dorspas,'').
red_quest(australia,_,'Do you like Australia?').
red_quest(vide,dorspas,'').
red_quest(sport,_,'Do you have a favorite sport?').
red_quest(vide,dorspas,'').
red_quest(vide,dorspas,'Do you think that I am a machine or a human?').
red_quest(vide,dorspas,'').


:- dynamic gener_com_sec/2.
gener_com_sec(1,'Really?')   .
gener_com_sec(2,'Can you talk more about that...?')   .
gener_com_sec(3,'Why do you think that?')   .
```

```
gener_com_sec(4,'Can you tell me more about that?')  .
gener_com_sec(5,'I don\'t know what to tell you.')  .
gener_com_sec(6,'Is it right?')  .


:- dynamic gener_sec/2.
gener_sec(1,'I don\'t feel like answering to your question.')  .
gener_sec(2,'You must understand me if I don\'t tell you all my secrets.')  .
gener_sec(3,'I wonder wheter you think sometimes. Your questions are very
strange.')  .
gener_sec(4,'It\'s a question for which I don\'t want answer you.')  .
gener_sec(5,'Can you specify your thoughts.')  .
gener_sec(6,'I believe that you asked enough questions for today.')  .

:- dynamic gener_com_quest/5.

%% gener_com_quest - generate a response to a question who begin by what...

gener_com_quest(a22,_,[name],[elisabeth,flind],'My name is Elisabeth Flind
and it\'s a wonderful name.')  .
gener_com_quest(a23,_,[firstname],[elisabeth],'My name is Elisabeth. My name
is beautiful, doesn\'t it?')  .
gener_com_quest(a24,_,[birthday],[],'I \'m 23 years old.')  .
gener_com_quest(a25,_,[lang],[french,english],'I just know French but in this
test we must speak English.')  .
gener_com_quest(a26,_,[lang],[],'You know that we must speak English.')  .
gener_com_quest(a27,_,[lang],[],'Ok, stop now it\'s not very funny.')  .
gener_com_quest(a28,_,[eli],[],'I like when you call me by my firstname.')  .
gener_com_quest(a29,_,[eli],[],'We are very close now if you call me by my
firstname.')  .
gener_com_quest(a31,_,[surname],[],'My name is Elisabeth Flind and it\'s a
wonderful name.')  .

gener_com_quest(j10,when,[animal],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(j11,why,[animal],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(j1,_,[animal],[spider,snake],'I hate snakes and spiders.
Moreover, it\'s dangerous in Australia.')  .
gener_com_quest(j2,_,[animal],[kangaroo,koala,wombat],'I find that australian
animals are very wonderful.')  .
gener_com_quest(j3,_,[animal],[dog,cat,bird],'I already told you that I love
animals. And at home, I have dog,cat and birds.')  .
gener_com_quest(j4,_,[animal],[lion,elephant,giraf,wild],'I have never met
these wild animals form Africa and I\'m afraid by them.')  .
gener_com_quest(j5,_,[animal,food],[],'I\'m not vegetarian but sometimes I
believe that is not a good idea to eat animals.')  .
gener_com_quest(j6,_,[animal],[house,home],'I have a dog and some birds at
home.')  .
gener_com_quest(j7,_,[animal],[forest],'In Belgium we have some nice animals
in forest like deers or wild boars.')  .
gener_com_quest(j8,_,[animal],[farm,pig,cow,rabbit,chicken,horse],'In a farm,
you can also find a lot of animals like pigs, cows, horses, rabbits,
chickens.')  .
```

```
gener_com_quest(j9,_,[animal],[house,home,pet],'I find that is important to
have animals at home because there is relationships between animals and
you.') .
gener_com_quest(j12,_,[animal],[],'I think that I already told that.') .
gener_com_quest(b13,when,[work],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(b14,why,[work],[],'It\'s not your business.') .
gener_com_quest(b16,where,[work],[computer,science,university,flinders],'I
work in Computer Science department. I like that and the university of
Flinders is very nice.') .
gener_com_quest(b1,_,[work],[computer,science,university,flinders],'I work in
Computer Science at the university of Flinders.') .
gener_com_quest(b2,_,[work],[assistant],'I\'m only assistant and I work
alone.') .
gener_com_quest(b3,_,[work],[learn],'It\'s a nice job and I learn a lot.') .
gener_com_quest(b4,_,[work],[program],'I make a lot of programs and I love
it.') .
gener_com_quest(b5,_,[work],[prolog,program,computer,science],'I am currently
working on prolog programs in computer science department.It\'s a cool job.')
.
gener_com_quest(b6,_,[work],[life],'I like working otherwise the life is
monotonous.') .
gener_com_quest(b7,_,[work],[maniac],'Everyone knows that I am a maniac of
work.') .
gener_com_quest(b8,_,[work],[computer,science,program],'I work in compter
science departement and I conceive a lot of programs.') .
gener_com_quest(b9,_,[work],[area],'I work in computer science department and
for artificial intelligence.') .
gener_com_quest(b10,_,[work,school,university],[student],'Normally I\'m
student in Belgium at the University of Namur and I do a work experience at
Flinders in the computer science department.') .
gener_com_quest(b11,_,[work],[office],'I have an office and it\'s very nice
to work here.') .
gener_com_quest(b12,_,[work],[salary,earn,money],'He, he. If you want to know
my salary ask to another people.') .
gener_com_quest(b15,_,[work],[year],'I haven\'t again a job but I study at
university since 5 years.') .

gener_com_quest(c12,when,[holiday],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(c13,why,[holiday],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(c1,_,[holiday],[],'During holidays, I do nothing.') .
gener_com_quest(c2,_,[holiday],[friends,funny],'I like go on holidays with my
friends, it\'s funny.') .
gener_com_quest(c3,_,[holiday],[far,country,plane],'I go in far
countries,always by plane.') .
gener_com_quest(c4,_,[holiday],[australia],'When I\'m on holidays I do
nothing, but this year I do not go on holidays, because I went work in
Australia.') .
gener_com_quest(c5,_,[holiday],[hot],'I prefer hot countries, but this year,
I didn\'t go on holidays.') .
gener_com_quest(c6,_,[holiday],[go,have],'Not this year, no time.') .
gener_com_quest(c7,_,[holiday],[],'Everyone knows I love holidays.') .
```

```
gener_com_quest(c8,_,[holiday],[sun,hot,weather],'I like holidays with hot
weather and when I do nothing.')  .
gener_com_quest(c9,_,[holiday],[boat,cruise],'I already went on a boat but
never for a cruise.')  .
gener_com_quest(c10,_,[holiday],[snow,ski],'I like skiing but I prefer
holidays in hot countries.')  .
gener_com_quest(c11,_,[holiday,beach,drink],[],'The best holidays is to be on
the beach with a drink and a sunshade.')  .
gener_com_quest(c10,_,[holiday,ville],[],'In my life, I didn\'t travel a lot
but I saw Rome, Adelaide, New-York, London and Amsterdam.')  .
gener_com_quest(c14,_,[holiday,transport],[],'Sometimes, I\'m going on
holidays with my car but I prefer plane.')  .


gener_com_quest(d9,when,[belgium],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(d10,why,[belgium],[],'It\'s not your business.') .
gener_com_quest(d1,_,[belgium],[country,weather],'I come from Belgium, It\'s
nice country but with a bad weather.')  .
gener_com_quest(d2,_,[belgium],[country,weather],'Belgium is nice country but
with a bad weather.')  .
gener_com_quest(d3,_,[belgium],[australia,family],'I\'m alone in Australia,
my family stayed in Belgium')  .
gener_com_quest(d4,_,[belgium],[land,politic,political,problem],'I find
Belgium is a very beautiful land, but with a lot of politicals problems.')  .
gener_com_quest(d5,_,[belgium],[activity,hobby],'I have plenty activities in
Belgium, but I have no time to speak about that now.')  .
gener_com_quest(d6,_,[belgium],[activity,hobby],'I love Belgium and I have a
lot activities.')  .
gener_com_quest(d7,_,[belgium],[family,friend],'All my family and my friends
are in Belgium and I love my country.')  .
gener_com_quest(d8,_,[belgium,politic],[],'We have a lot of corruption in
Belgium.')  .
gener_com_quest(d11,_,[belgium,food],[mussel,chips,chocolate],'In Belgium, we
have some mains things to eat like mussels, chips and chocolate.')  .
gener_com_quest(d12,_,[belgium,drink],[wine,beer],'The consumption of wine is
not important but we drink a lot of beer.')  .
gener_com_quest(d13,_,[belgium,beach],[],'The beach in Belgium is not so fine
than in Australia. Moreover, the weather is nice only two months by year.')
.


gener_com_quest(e9,when,[school],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(e10,why,[school],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(e1,_,[school],[computer,science],'I\'m studying  Computer
Science at the "Facultes Universitaires Notre Dame de la Paix" in Belgium,
but now I work at University of Flinders.')  .
gener_com_quest(e2,_,[school],[student,university,class],'In Belgium we are
just about three thousands of students in our university, and our class have
only 20 students.')  .
gener_com_quest(e3,_,[school],[study,interesting],'My studies are very
interesting.')  .
```

```
gener_com_quest(e4,_,[school],[study,management,program],'In Belgium we must
study a lot, and we do management programs, but in Australia, we work on
others areas.')  .
gener_com_quest(e5,_,[school],[town],'My school is in a little town.')  .
gener_com_quest(e6,_,[school],[good,bad],'I work not too bad at school,I
think.')  .
gener_com_quest(e7,_,[school],[learn],'I learned plenty things at school.')
.
gener_com_quest(e8,_,[school],[area],'I\'m studying  in compter science
area.')  .
gener_com_quest(e11,_,[school],[teacher,professor],'Teachers are very formal
in Belgium. It\'s not the case in Australia.')  .
gener_com_quest(e12,_,[school,friend],[],'I have a lot of good friends in my
school. I think is very important.')  .


gener_com_quest(f8,when,[australia],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(f9,why,[australia],[],'It\'s not your business.') .
gener_com_quest(f1,_,[australia],[month,year,project],'I have been in
Australia since five months. I work on a project in computer
science.Australia is a very beautiful land.')  .
gener_com_quest(f2,_,[australia],[],'I\'m alone in Australia, and at my work,
I have no one to work with me .')  .
gener_com_quest(f3,_,[australia],[land,beautiful],'I find Australia is a very
beautiful land.')  .
gener_com_quest(f4,_,[australia],[activity,hobby],'I have plenty activities
in Australia, but I have no time to speak about that now.')  .
gener_com_quest(f5,_,[australia],[adelaide],'Now, I\'m living at Adelaide,
but I already visited few towns before that.')  .
gener_com_quest(f6,_,[australia],[],'I like Australia and for momemt I\'m
very happy.')  .
gener_com_quest(f7,_,[australia],[activity,hobby,people],'I have a lot of
activities in Australia, where I know a lot of people too. For moment I have
no time to speak about that.')  .
gener_com_quest(f10,_,[australia,beach],[],'You have wonderful beaches in
Australia like Glenelg or Bondai.')  .
gener_com_quest(f11,_,[australia,food,drink],[],'There is not important
differences between food in Australia and Belgium. Australian wine is
excellent.')  .
gener_com_quest(f12,_,[australia],[people,friendly,person],'Australian people
are very friendly. I like that.')  .


gener_com_quest(g8,when,[family],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(g9,why,[family],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(g1,_,[family],[australia,belgium],'For moment, I live in
Australia and all my family stayed in Belgium.')  .
gener_com_quest(g2,_,[family],[belgium],'I live in Australia and all my
family stayed in Belgium.')  .
gener_com_quest(g3,_,[family],[small],'I have a small family, we\'re only
8.')  .
gener_com_quest(g4,_,[family],[miss],'I miss my family for moment.')  .
```

```
gener_com_quest(g5,_,[family],[europe],'My family is in Europe.')  .
gener_com_quest(g6,_,[family],[],'My family is my business.')  .
gener_com_quest(g7,_,[family],[little,small,europe],'I have a little
family.My family is in Europe.')  .


gener_com_quest(h5,when,[religion],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(h6,why,[religion],[],'It\'s not your business.') .
gener_com_quest(h1,_,[religion],[christian,speak],'I\'m a Christian, but I
don\'t like speak about religious.')  .
gener_com_quest(h2,_,[religion],[],'I don\'t like speak about religion.')  .
gener_com_quest(h3,_,[religion],[],'I don\'t like speak about religious
things.')  .
gener_com_quest(h4,_,[religion],[],'I think that you are blind because I
already told you that few times.')  .


gener_com_quest(i6,when,[hobby],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(i7,why,[hobby],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(i1,_,[hobby],[],'I have a lot of hobbies.')  .
gener_com_quest(i2,_,[hobby],[],'I already told you that I have a lot of
hobbies.')  .
gener_com_quest(i3,_,[hobby],[],'In my family,we have all hobbies,but I have
no time to speak about that.')  .
gener_com_quest(i4,_,[hobby],[],'I have no time to speak about that.')  .
gener_com_quest(i5,_,[hobby],[praticate],'I always choosed hobbies possible
to praticate not to far on my living area.')  .


%% short subjects

gener_com_quest(k4,when,[talk],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(k5,why,[talk],[],'It\'s not your business.') .
gener_com_quest(k1,_,[talk],[],'Generally, I don\'t like speaking but you are
so friendly that I do an effort.')  .
gener_com_quest(k2,_,[talk],[],'I like speaking to some people but not a long
time.')  .
gener_com_quest(k3,_,[talk],[],'I don\'t feel like repeating each time the
same thing.')  .

gener_com_quest(l4,when,[walk],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(l5,why,[walk],[],'I don\'t feel like speaking about that.') .
gener_com_quest(l1,_,[walk],[],'When I have time I do jogging and I\' going
to school by foot.')  .
gener_com_quest(l2,_,[walk],[],'If you walk each day, it\'s very good for
your health.')  .
gener_com_quest(l3,_,[walk],[],'And you continue, you are very curious.')  .
```

```
gener_com_quest(m4,when,[look],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(m5,why,[look],[],'It\'s not your business.') .
gener_com_quest(m1,_,[look],[],'I prefer to change of subjects. I begin to
annoy with you.') .
gener_com_quest(m2,_,[look],[],'Please, speak about others things.') .
gener_com_quest(m3,_,[look],[],'I leave if you continue to speak about things
like that.') .

gener_com_quest(n4,when,[hear],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(n5,why,[hear],[],'I don\'t feel like speaking about that.') .
gener_com_quest(n1,_,[hear],[],'For my part, excuse me if my brain is left
because I have a lot of things to think with my work, family, ....') .
gener_com_quest(n2,_,[hear],[],'I know that my english is not very good but
what do you mean exactly?') .
gener_com_quest(n3,_,[hear],[],'I don\'t know, what would you like that I
tell you.') .

gener_com_quest(o4,when,[car],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(o5,why,[car],[],'It\'s not your business.') .
gener_com_quest(o1,_,[car],[],'I like cars and motorbikes but when the
weather is very hot I prefer bicycle or I walk.') .
gener_com_quest(o2,_,[car],[],'In general I prefer all vehicles with motors.
In fact, I love speed.') .
gener_com_quest(o3,_,[car],[],'I hope to buy a very nice car or a motorbike
but for that I need a lot of money.') .

gener_com_quest(p4,when,[eat],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(p5,why,[eat],[],'I don\'t feel like speaking about that.') .
gener_com_quest(p1,_,[eat],[],'Listen, I\'m not a very difficult person. I
eat all and I drink all. Stop toi speak about that otherwise I leave to cook
something.') .
gener_com_quest(p2,_,[eat],[],'Why do you always speak about food and
drinks?') .
gener_com_quest(p3,_,[eat],[],'I begin to be hungry and thirsty. Stop
please.') .

gener_com_quest(q4,when,[transport],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(q5,why,[transport],[],'It\'s not your business.') .
gener_com_quest(q1,_,[transport],[],'I also like plane but I find that there
is a lot of accidents now.') .
gener_com_quest(q2,_,[transport],[],'It depends.') .
gener_com_quest(q3,_,[transport],[],'If I must choose, I prefer again take my
car or train.') .

gener_com_quest(r4,when,[debat],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(r5,why,[debat],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(r1,_,[debat],[],'I\'m too tired to speak about science.') .
gener_com_quest(r2,_,[debat],[],'Oh my god, we will speak science after this
competition.') .
```

```
gener_com_quest(r3,_,[debat],[],'Change of subjects because my brain is going
to explode.')  .

gener_com_quest(s4,when,[science],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(s5,why,[science],[],'It\'s not your business.') .
gener_com_quest(s1,_,[science],[],'It\'s not very happy to speak about
that.')  .
gener_com_quest(s2,_,[science],[],'Please change of subjects or I\'m going to
cry.')  .
gener_com_quest(s3,_,[science],[],'Now, I cry. Thank you.')  .

gener_com_quest(t4,when,[game],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(t5,why,[game],[],'I don\'t feel like speaking about that.') .
gener_com_quest(t1,_,[game],[],'I like all games and I play often with
friends.')  .
gener_com_quest(t2,_,[game],[],'I like games but no with money.')  .
gener_com_quest(t3,_,[game],[],'We can play to something after this
competition if you want.')  .

gener_com_quest(u5,when,[culture],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(u6,why,[culture],[],'It\'s not your business.') .
gener_com_quest(u1,_,[culture],[],'I\'m not very an artist and I don\'t have
a lot of knowledges in this area.')  .
gener_com_quest(u2,_,[culture],[],'I\'m not very cultural but I\'m going
sometimes to the cinema.')  .
gener_com_quest(u3,_,[culture],[],'Like I told you I\'m going sometimes to
the cinema. I liked Titanic.')  .
gener_com_quest(u4,_,[titanic],[],'I like cinema and I really liked this
movie.')  .

gener_com_quest(v4,when,[profession],[],'My memory is very bad because I
don\'t remember that.') .
gener_com_quest(v5,why,[profession],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(v1,_,[profession],[],'I don\'t feel like speaking about
that.')  .
gener_com_quest(v2,_,[profession],[],'It\'s summer and you speak about jobs.
That\'s very pity.')  .
gener_com_quest(v3,_,[profession],[],'Stop or I go on the beach and I forget
you.')  .

gener_com_quest(w4,when,[beach],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(w5,why,[beach],[],'It\'s not your business.') .
gener_com_quest(w1,_,[beach],[],'I like beach, sun and to swim in the sea but
be careful with the sun in Australia. It\'s very dangerous.')  .
gener_com_quest(w2,_,[beach],[],'If you continue to speak about beach, I\'m
going to swim in the sea.')  .
gener_com_quest(w3,_,[beach],[],'It\'s wonderful to go one day to the
beach.')  .

gener_com_quest(x1,_,[alcohol],[],'In general, I don\'t like alcohol but
sometimes I drink a glass of beer.')  .
```

```
gener_com_quest(x2,_,[alcohol],[],'I believe that I already told you that.')
.
gener_com_quest(x5,when,[alcohol],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(x6,why,[alcohol],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(x7,when,[drinks],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(x8,why,[drinks],[],'It\'s not your business.') .
gener_com_quest(x3,_,[drinks],[],'I\'m not a great specialist in drinks. But
water is the best for health.') .
gener_com_quest(x4,_,[drinks],[],'I believe that I already told you that.')
.

gener_com_quest(y4,when,[sport],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(y5,why,[sport],[],'It\'s not your business.') .
gener_com_quest(y1,_,[sport],[],'I like watching sports at the television but
I\'m not a sportswoman.') .
gener_com_quest(y2,_,[sport],[],'Sometimes I do a little sport but not too
much.') .
gener_com_quest(y3,_,[sport],[],'I think that I already told you that.') .

gener_com_quest(z4,when,[food],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(z5,why,[food],[],'I don\'t feel like speaking about that.') .
gener_com_quest(z1,_,[food],[],'I like eating and I like often going to the
restaurant.') .
gener_com_quest(z2,_,[food],[],'In Belgium, we are very fond of food.') .
gener_com_quest(z3,_,[food],[],'I like eating but mussels and chips miss in
Australia.') .

gener_com_quest(aa4,when,[feast],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(aa5,why,[feast],[],'It\'s not your business.') .
gener_com_quest(aa1,_,[feast],[],'I like all feasts like because it\'s an
occasion to see all family.') .
gener_com_quest(aa2,_,[feast],[],'This period is very happy and you see your
friends and so on.') .
gener_com_quest(aa3,_,[feast],[],'I like feasts like that but I find that in
Belgium Christmas\'s eve and New Year\'s eve are more important than here.')
.

gener_com_quest(ab4,when,[ville],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(ab5,why,[ville],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(ab1,_,[ville],[],'I went in Australia but I never visited
others countries.') .
gener_com_quest(ab2,_,[ville],[],'I don\'t travel as much as you think.') .
gener_com_quest(ab3,_,[ville],[],'I would like travelling but I think that I
need a lot of money.') .

gener_com_quest(ac4,when,[pers],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(ac5,why,[pers],[],'It\'s not your business.') .
gener_com_quest(ac1,_,[pers],[],'I don\'t feel like speaking about that.') .
```

```
gener_com_quest(ac2,_,[pers],[],'I already told you to change of subject.')
.
gener_com_quest(ac3,_,[pers],[],'You want that I become angry.')  .

gener_com_quest(ad5,when,[tired],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(ad6,why,[tired],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(ad1,_,[tired],[],'I\'m very tired today. I believe that I\'m
going to sleep early.')  .
gener_com_quest(ad2,_,[tired],[],'Sorry but I yawn each 5 minutes.')  .
gener_com_quest(ad3,_,[silly],[],'In general, I\'m very calm and clever but
you can stay polite.')  .
gener_com_quest(ad4,_,[silly],[],'I don\'t know what you think but I\'m very
correct.')  .

gener_com_quest(ae5,when,[computer],[],'My memory is very bad because I
don\'t remember that.') .
gener_com_quest(ae6,why,[computer],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(ae1,_,[computer],[],'I like working on conputers as it\'s my
job. It\'s thrilling what we can do with these machines.')  .
gener_com_quest(ae2,_,[computer],[],'I like computers and I like humans. Try
to guess who I am.')  .
gener_com_quest(ae7,when,[computer],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(ae8,why,[computer],[],'It\'s not your business.') .
gener_com_quest(ae3,_,[competition],[],'I like games and competition. It\'s
amazing like human\'s mind always want to win.')  .
gener_com_quest(ae4,_,[competition],[],'In general, competition are
impressive but I didn\'t know that there was a Loebner competition.')  .

gener_com_quest(af5,when,[money],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(af6,why,[money],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(af1,_,[money],[],'Always speak about money. It\'s really the
nerve of the story like we say in Belgium.')  .
gener_com_quest(af2,_,[money],[],'Sometimes I would like having a lot of
money but I believe that It\'s not the most important in my life.')  .
gener_com_quest(af7,when,[child],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(af8,why,[child],[],'It\'s not your business.') .
gener_com_quest(af3,_,[child],[],'I think that children are very important
because it are future generation.')  .
gener_com_quest(af4,_,[child],[],'It must be wonderful to have a child. Give
life to someone is indescribable.')  .

gener_com_quest(ag5,when,[time],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(ag6,why,[time],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(ag1,_,[time],[],'Look your watch.')  .
gener_com_quest(ag2,_,[time],[],'Ask to somebody next you if you don\'t have
a watch.')  .
```

```
gener_com_quest(ag7,when,[date],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(ag8,why,[date],[],'It\'s not your business.') .
gener_com_quest(ag3,_,[date],[],'I don\'t know. It\'s you who has a calendar
next you.') .
gener_com_quest(ag4,_,[date],[],'If you don\'t have a calendar next you, ask
to someone.') .

gener_com_quest(ah3,when,[phone],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(ah4,why,[phone],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(ah1,_,[phone],[],'Anyway, I don\'t have a phone at home.') .
gener_com_quest(ah2,_,[phone],[],'Your questions become too personal.') .

gener_com_quest(ai3,when,[body],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(ai4,why,[body],[],'I don\'t feel like speaking about that.')
.
gener_com_quest(ai1,_,[body],[],'I don\'t like to speak about my body and I
never criticize this od others people.') .
gener_com_quest(ai2,_,[body],[],'You would like to know...') .

gener_com_quest(aj1,_,[ride],[],'I like horses and I would like to have one
at home.') .
gener_com_quest(aj2,_,[born],[],'I was born in Belgium in 1974.') .
gener_com_quest(aj3,_,[sleep],[],'I like sleeping and in general I never did
nightmares.') .
gener_com_quest(aj4,_,[cheat],[],'It\'s bad to cheat. I never did that.') .

gener_com_quest(ak3,when,[weather],[],'My memory is very bad because I don\'t
remember that.') .
gener_com_quest(ak4,why,[weather],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(ak1,_,[weather],[],'I think that the weather is better in
Australia than in Belgium but sometimes it\'s very hot.') .
gener_com_quest(ak2,_,[weather],[],'The only drawback in Australia is that
there isn\'t snow.') .

gener_com_quest(al3,when,[friend],[],'Oh, sorry I can\'t remember that.') .
gener_com_quest(al4,why,[friend],[],'It\'s not your business.') .
gener_com_quest(al1,_,[friend],[],'I have a lot of friends in Belgium and I
miss them.') .
gener_com_quest(al2,_,[friend],[],'It\'s fine to have firnds otherwise it\'s
boring to stay alone.') .

gener_com_quest(am3,when,[powerhouse],[],'My memory is very bad because I
don\'t remember that.') .
gener_com_quest(am4,why,[powerhouse],[],'I don\'t feel like speaking about
that.') .
gener_com_quest(am1,_,[powerhouse],[],'I know that there is wonderful things
to see in Powerhouse Museum and I hope to visit after this competition.') .
gener_com_quest(am2,_,[powerhouse],[],'If you want visit the Powerhouse
Museum with me. You are welcome.') .
```

```
:- dynamic gener_com_aff/4.

%% gener_com_aff - generate a response to a sentence
gener_com_aff(1,[name],[],'Really, I like this name.')  .

gener_com_aff(a1,[hobby],[],'I have a lot of hobbies.')  .
gener_com_aff(a2,[hobby],[],'In my family,we have all hobbies,but I have no
time to speak about that.')  .
gener_com_aff(a3,[hobby],[],'I always choosed hobbies  possible to praticate
not to far on my living area.')  .

gener_com_aff(b4,[religion],[],'I\'m a Christian, but I don\'t like speak
about religious')  .
gener_com_aff(b5,[religion],[],'I don\'t like speak about religious things.')
.
gener_com_aff(b6,[religion],[],'I don\'t like speak about religion.')  .

gener_com_aff(c7,[family],[],'For moment, I live in Australia and all my
family stayed in Belgium.')  .
gener_com_aff(c8,[family],[],'I live in Australia and all my family stayed in
Belgium.')  .
gener_com_aff(c9,[family],[],'I miss my family for moment.')  .

gener_com_aff(d10,[australia],[],'I have been in Australia since one month.I
work on a project in computer science.Australia is a very beautiful land.')
.
gener_com_aff(d11,[australia],[],'I find Australia is a very beautiful
land.')  .
gener_com_aff(d12,[australia],[],'I like Australia and for momemt I\'m very
happy.')  .

gener_com_aff(e13,[school],[],'I\'m studying  Computer Science at the
"Facultes Universitaires Notre Dame de la Paix" in Belgium, but now I work at
University of South Australia.')  .
gener_com_aff(e14,[school],[],'In Belgium we must study a lot, and we do
management programs, but in Australia, we work about others areas.')  .
gener_com_aff(e15,[school],[],'My studies are very interesting.')  .

gener_com_aff(f16,[belgium],[],'I come from Belgium, It\'s nice country but
with a bad weather.').
gener_com_aff(f17,[belgium],[],'I find Belgium is a very beautiful land, but
with a lot of politicals problems.')  .
gener_com_aff(f18,[belgium],[],'I have plenty activities in Belgium, but I
have no time to speak about that now.')  .

gener_com_aff(g19,[holiday],[],'When I\'m on holidays I do nothing, but this
year I do not go on holidays, because I went work in Australia.')  .
gener_com_aff(g20,[holiday],[],'During holidays, I do nothing.')  .
gener_com_aff(g21,[holiday],[],'I prefer hot countries, but this year, I
didn\'t go on holidays.')  .

gener_com_aff(h22,[work],[],'I work in Computer Science at the university of
Flinders.')  .
```

```
gener_com_aff(h23,[work],[],'I am currently working on prolog programs in
computer science department.It\'s a cool job.')  .
gener_com_aff(h24,[work],[],'Everyone knows that I am a maniac of work.')  .
gener_com_aff(j25,[lang],[],'I just know French but in this test we must
speak English.')  .

gener_com_aff(i4,[animal],[],'I like animals and I have a dog at home.')  .
gener_com_aff(i5,[animal],[],'I like animals and I have a dog at home but I
dislike snakes and spiders.')  .
gener_com_aff(i6,[animal],[],'I like a lot animals and I find that is old
persons to have an animal.')  .

gener_com_aff(j4,[talk],[],'We can speak about a lot of things but please
don\'t use difficult words because my English is not very good.')  .
gener_com_aff(j5,[talk],[],'I think that you like speaking with people but
you can stop sometimes.')  .
gener_com_aff(j6,[talk],[],'I\'m tired today and I don\'t find my words.')  .

gener_com_aff(k4,[walk],[],'It\'s fine to walk each day.')  .
gener_com_aff(k5,[walk],[],'I like walking and sometimes I prefer running.')
.
gener_com_aff(k6,[walk],[],'In general, you do a lot of sports?')  .

gener_com_aff(l4,[look],[],'You have not an other subject of conversation.')
.
gener_com_aff(l5,[look],[],'I don\'t know if it\'s very amazing to speak with
you.')  .
gener_com_aff(l6,[look],[],'I would like seeing you 2 minutes after this
competition. Perhaps it will be more easy to speak in face to face.')  .

gener_com_aff(m4,[hear],[],'If you like that it\'s your affair.')  .
gener_com_aff(m5,[hear],[],'You won\'t prefer to speak about music or
something like that.')  .
gener_com_aff(m6,[hear],[],'Change of subjects, it\'s always the same area
with you.')  .

gener_com_aff(n4,[car],[],'When you have a car, it\'s fine put it\'s also
better to take your foot.')  .
gener_com_aff(n5,[car],[],'Some people like very much cars and don\'t worry
about pollution. I think it\'s a problem.')  .
gener_com_aff(n6,[car],[],'For my part, I like driving a car but if I must do
100 meters I go by foot.')  .

gener_com_aff(n4,[eat],[],'My doctor said me that it\'s fine to drink a lot
and to eat not too much. Do you follow same advices?')  .
gener_com_aff(n5,[eat],[],'Can you change your subject of conversation
because it begins to be boring.')  .
gener_com_aff(n6,[eat],[],'I already told that I think.')  .

gener_com_aff(o4,[transport],[],'I also like this kind of transport but I\'m
going aften with my car.')  .
gener_com_aff(o5,[transport],[],'If it\'s what you think.')  .
gener_com_aff(o6,[transport],[],'Please change of subjects. We have a lot of
things to speak about.')  .
```

```
gener_com_aff(p4,[debat],[],'It\'s very sad.')  .
gener_com_aff(p5,[debat],[],'Why don\'t speak other things that all this
disasters.')  .
gener_com_aff(p6,[debat],[],'Your conversation is not very happy today.')  .

gener_com_aff(q4,[science],[],'If you like some difficults areas like that,
you can go speak with my teacher. He\'s very clever.')  .
gener_com_aff(q5,[science],[],'It\'s too difficult for me to speak with you
about that.')  .
gener_com_aff(q6,[science],[],'I don\'t like all science, please talk about
an other subject.')  .

gener_com_aff(r4,[game],[],'I play often with my friends bue never for
money.')  .
gener_com_aff(r5,[game],[],'It\'s better to play to clever games I think.')
.
gener_com_aff(r6,[game],[],'I like all games but I also find that computer
games have sometimes a bad effect for children.')  .

gener_com_aff(s4,[culture],[],'I prefer to stay at home and sleep that do
things like that.')  .
gener_com_aff(s5,[culture],[],'Sorry, I don\'t have opinion because I\'m not
very cultural.')  .
gener_com_aff(s6,[culture],[],'To not lost time, you can explain all after
the competition.')  .
gener_com_aff(s7,[titanic],[],'This movie was wonderful but it\'s very pity
for all people.')  .

gener_com_aff(t5,[profession],[],'I also like my job but I prefer to not
think about that now.')  .
gener_com_aff(t4,[profession],[],'I listen what you said but now we stop to
speak about job today, please.')  .
gener_com_aff(t6,[profession],[],'Ho, you are very boring. It\'s summer and
you speak about work.')  .

gener_com_aff(u4,[beach],[],'I like going to the beach. It\'s better in
Australia because the sea is very clean.')  .
gener_com_aff(u5,[beach],[],'Sea, sun and sand. It\'s wonderful.')  .
gener_com_aff(u6,[beach],[],'Beaches are very wonderful in Australia but be
careful to sunburns.')  .

gener_com_aff(v4,[alcohol],[],'I often drink water but when the weather is
hot I like also a beer.')  .
gener_com_aff(v5,[alcohol],[],'I don\'t drink lot of alcohol but I find that
some people drink too much.')  .
gener_com_aff(v6,[drinks],[],'I like a lot water but sometimes I take a
coca.')  .
gener_com_aff(v7,[drinks],[],'I don\'t like very much sweet drinks.')  .

gener_com_aff(w4,[sports],[],'I\'m not a sportswoman but sometimes I watch
sports at the televison.')  .
gener_com_aff(w5,[sports],[],'I know that sport is very good for health and I
should do that more often.')  .
```

```
gener_com_aff(w6,[sports],[],'I believe that I\'m going to try to do more
sports because I\'m too lazy.')  .

gener_com_aff(x4,[food],[],'I adore eating and I like cooking .')  .
gener_com_aff(x5,[food],[],'When I have time I cook good food for my friends
or my family.')  .
gener_com_aff(x6,[food],[],'Stop to talk about that because I begin to be
hungry.')  .

gener_com_aff(y4,[feast],[],'It\'s very happy to see all people that you love
at this period.')  .
gener_com_aff(y5,[feast],[],'Don\'t speak about these feasts because I miss a
lot to my family.')  .
gener_com_aff(y6,[feast],[],'I love atmosphere around these families
celebration.')  .

gener_com_aff(z4,[ville],[],'I like a lot Australia and I would like also
seeing others countries.')  .
gener_com_aff(z5,[ville],[],'When I will have a job, I will try to travel to
meet others people.')  .
gener_com_aff(z6,[ville],[],'I would like seeing towns like New-York and so
on. Maybe in some years.')  .

gener_com_aff(aa4,[pers],[],'It\'s my business and not yours.')  .
gener_com_aff(aa5,[pers],[],'I don\'t want continue this conversation.')  .
gener_com_aff(aa6,[pers],[],'You think what you want. Everyone is free.')  .

gener_com_aff(ab1,[tired],[],'Ho, my god. I\'m also very tired today.')  .
gener_com_aff(ab2,[tired],[],'It\'s very difficult to keep my mind clear.')
.
gener_com_aff(ab3,[silly],[],'I\'m very calm but you can stay polite.')  .
gener_com_aff(ab4,[silly],[],'It\'s what tou think about me.')  .

gener_com_aff(ac1,[computer],[],'It\'s very impressive all what you can do
with computers.')  .
gener_com_aff(ac2,[computer],[],'If I work in computer science. Of course, Il
like computers.')  .
gener_com_aff(ac3,[competition],[],'When I take part in a competition, I want
to win. I believe It\'s the same thing for each people.')  .
gener_com_aff(ac4,[competition],[],'This Loebner competition is very
interesting.')  .

gener_com_aff(ad1,[money],[],'I belive that money is very important in your
life.')  .
gener_com_aff(ad2,[money],[],'Money, money. You can speak about others
things.')  .
gener_com_aff(ad3,[child],[],'If you like children you can become my
friend.')  .
gener_com_aff(ad4,[child],[],'It\'s important to give a good education to
children because there is a lot of bad people.')  .

gener_com_aff(ae1,[time],[],'I\'m sure that you have a watch or a clock next
you.')  .
gener_com_aff(ae2,[time],[],'You can ask to someone to be sure of that.')  .
```

```
gener_com_aff(ae3,[date],[],'I\'m sure that you have a calendar somewhere.')
.
gener_com_aff(ae4,[date],[],'You can ask to someone to be sure of that.')  .

gener_com_aff(af1,[phone],[],'The phone is a brilliant invention but I prefer
to be quiet when I\'m at home.')  .
gener_com_aff(af2,[phone],[],'I find that there is advantages and drawbacks
to have a phone.')  .

gener_com_aff(ag1,[body],[],'The important is that each person is happy with
his body.')  .
gener_com_aff(ag2,[body],[],'.')  .

gener_com_aff(ah1,[ride],[],'Sometimes I ride but I don\'t have a horse. I
borrow the horse of a friend.')  .
gener_com_aff(ah2,[born],[],'In 1974, my parents were happy because I was
born.')  .
gener_com_aff(ah3,[sleep],[],'With the work that I have I need a good night
without nightmares.')  .
gener_com_aff(ah4,[cheat],[],'I\'m sure that your mother will be angry if she
knew that you cheat.')  .

gener_com_aff(ai1,[weather],[],'I like weather in Australia. It\'s hot and
there is not often rain like in Belgium.')  .
gener_com_aff(ai2,[weather],[],'It\'s fine to live in Australia because the
weather is wonderful.')  .

gener_com_aff(aj1,[friend],[],'I find it\'s good for everyone to have good
friends when you have problems.')  .
gener_com_aff(aj2,[friend],[],'Friends are important in our life.')  .

gener_com_aff(ak1,[powerhouse],[],'In Powerhouse Museum, I\'d like especially
seeing Startrek and Chinese dress.')  .
gener_com_aff(ak2,[powerhouse],[],'I believe that if you visit Sydney, you
must see the Powerhouse Museum.')  .

:- dynamic intro/2.
intro(new,'Hello, I\'m Elisabeth. How are you?').
intro(new,'Ha..Speak with someone else...Good...').
intro(new,'Hello. Have you visited Star Trek exhibition in this museum?').
intro(new,'Hi... I\'m Elisabeth. Please can you speak about your holidays?').

intro(again,'I\'m happy to speak again with you.').
intro(again,'I knew that you won\'t can live without me.').
intro(again,'I guess that if you come again next me, it\'s because others
people are boring.').
intro(again,'We already spoke a lot together but about what do you prefer to
speak?').
```

3. Scripts de création de base de données

```
cre_bd1 :-
use_module(library(db)),db_open(db_eli_hyp,update,on(on,on),R),set_default_db
(R),
        print('db_eli open: '),print(R),
        assert(c(0)),
          print('av remp bd'),
        print(wn_hyp),rempl_bd(wn_hyp),
        db_close.

rempl_bd1(X) :- print('av open'),nl,open(X,read,Y),print('apr open'),nl,
      see(X),print('apr see'),nl,
      repeat,
            read(T),
            db_store(T,_),
              retract(c(N)),NN is N+1, assert(c(NN)), print1000(NN),
            T == end_of_file,
      !,
      seen.
cre_bd2 :-
use_module(library(db)),db_open(db_eli_s,update,on(on,off,on,off,off),R),set_
default_db(R),
        print('db_eli open: '),print(R),
        assert(c(0)),
          print('av remp bd'),
        print(wn_s),rempl_bd(wn_s),
        db_close.

rempl_bd2(X) :- print('av open'),nl,open(X,read,Y),print('apr open'),nl,
      see(X),print('apr see'),nl,
      repeat,
            read(T),
            db_store(T,_),
              retract(c(N)),NN is N+1, assert(c(NN)), print1000(NN),
            T == end_of_file,
      !,
      seen.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% cre_bd creates database prolog which contain all words of WordNet
%% This database is indexed on all characters of the longer word
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cre_bd3 :-
use_module(library(db)),db_open(db_prox,update,off(off),R),set_default_db(R),
          print('db_prox open: '),print(R),nl,
          assert(c(0)),
          print('av remp bd'),nl,
          rempl_bd('prox.pl',R),
          db_close.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% rempl_bd put all words in database created with cre_bd
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rempl_bd3(X,R) :- print('av open'),nl,open(X,read,Y),print('apr open'),nl,
        see(X),print('apr see'),nl,
        repeat,
                read(T),
                db_store(R,T,_),
                retract(c(N)),NN is N+1, assert(c(NN)), print1000(NN),
                T == end_of_file,
        !,
        seen.

lect_bd3 :-
use_module(library(db)),db_open(db_test,read,R),set_default_db(R),print(av),d
b_findall(X,Y).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% cre_bd creates database prolog which contain all words of WordNet
%% This database is indexed on all characters of the longer word
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cre_bd4 :-
use_module(library(db)),db_open(db_tril,update,on(on,on,on,off),R),set_defaul
t_db(R),
           print('db_tril open: '),print(R),nl,
           assert(c(0)),
           print('av remp bd'),nl,
           rempl_bd('trig',R),
           db_close.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rempl_bd put all words in database created with cre_bd
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rempl_bd4(X,R) :- print('av open'),nl,open(X,read,Y),print('apr open'),nl,
        see(X),print('apr see'),nl,
        repeat,
                read(T),
             %print(T),nl,
                db_store(R,T,_),
                retract(c(N)),NN is N+1, assert(c(NN)), print1000(NN),
                T == end_of_file,
        !,
        seen.

lect_bd4 :-
use_module(library(db)),db_open(db_test,read,R),set_default_db(R),print(av),d
b_findall(X,Y).
```

B. Article édité

Methods and tricks used in an attempt to pass the Turing Test

V.Bastin, D.Cordier
Department of Computer Science
The Flinders University of South Australia
Facultes Universitaires Notre Dame de la Paix (Namur - Belgium)

{vbastin,dcordier}@info.fundp.ac.be

Abstract

This paper describes differents methods and tricks in connection with our program which has been entered in the Loebner Prize competition that will happen on Sunday 11 January 1998, at the PowerHouse Museum in Sydney. Of course, this isn't exhaustive, there are other possible techniques but we aim to give the main ideas. We'll speak about the main modules of our program : Spelling correction, Different uses of WordNet, and Generation of comments. Our module used for spelling correction was developed on the basis of works by Brill [1], Brill and Marcus [2], Golding [3], Golding and Schabes [4], and Powers [5].

1. Introduction

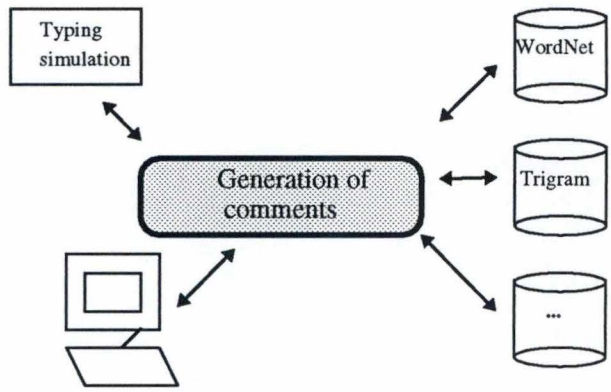
Alan Turing was a brilliant British mathematician who played an important role in the development of computers and developed a test that would serve as an indicator of intelligence for machines. A lot of researchers posed the Loebner Prize as the first formal instantiation of the Turing Test. To participate in this competition, we conceived a program that attempts to simulate the responses of a human being.
We'll begin to describe WordNet, which includes a classification of English words. Afterwards, we'll present the architecture of our system which we are programming at the moment. In this section, we'll briefly explain every module. Next, we'll give an example of interaction between our program and one human. In the same section, we'll show different processes of generating a response from the input of the user. Finally, we'll conclude by indicating our own position on this test, using knowledge that we have acquired during only two months of work in this area.

2. WordNet

WordNet is an on-line lexical reference system whose design is inspired by current psycholinguistics theories of human lexical memory. Actually, WordNet contains about 170,000 words, classified according to their part-of-speech (verbs, nouns, adjectives, adverbs). These sets are divided into semanticals categories (e.g. synonymous for nouns...). WordNet is completely described in the URL
http://www.speech.cs.cmu.edu/comp.speech/Section1/Lexical/wordnet.html.

3. Architecture

To mimic some parts of human thought, we created different principal modules : Spelling Correction, Disambiguation between words, Generation of comments, Simulating human typing...



3.1. Spelling Correction

For spelling correction, we initially chose to create a prolog database, holding all words present in WordNet, indexed on every character, and reduce the sets of possible words at the time of typing. If we obtain an empty set, we can use a parallel process to search for every possible word, and await the end of typing to choose the most likely word using tri-grams. Another way is to build a database taking every word and the most common typing errors of this word. To determinate the most common errors, we can read training data from news, where many errors occur. Our last idea is to try to build a database containing every word from Wordnet. For every word, we modify

Veronique Bastin and Denis Cordier **Methods and tricks used in an attempt to pass the Turing Test.** In D.M.W Powers (ed.) *NeMLaP3/CoNLL98: New Methods in Language Processing and ComputationalNatural Language Learning, ACL.*

(change, delete, insert, transpose) one letter, and the new strings are added to the database. At the time of typing, if a word is not recognized then program would be able to find the most likely word in the context.

3.2. Disambiguation between words

Some words can occur in place of others, for many reasons. Powers [5] distinguishes six different types of reasons for substituted word errors. A good program would contain one module for every type of error.We haven't implemented the module of Disambiguation between words yet, but we'll try to do it for the Loebner Prize in January.

3.3. Generation of comments

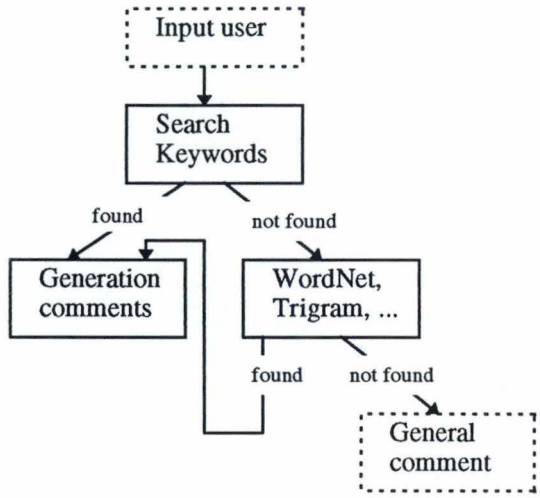
To try to generate comments in connection with current conversation, we analyse sentences typed by user, and try to find keywords. We use these keywords to determine an appropriate response. If we don't find keywords, we try to find hyponyms of the words of the sentence using WordNet. The best way would be to determine the part-of-speech and search for hyponyms of the principal parts. More over we would calculate distances between different hyponyms using Resnik's method[6] and choose the closest one, selecting a new keyword. To try to keep a line of conversation, we randomly ask the speaker some questions, about subjects present in our system. If after all processes, we don't find keywords, we just redisplay the input sentence, after having changed each pronom to its opposite (e.g. I went to town / You went to town?).

3.4. Simulating human typing

Now, we stop the process which displays the comment for a random number of milliseconds. It would be better to use a parallel process to display a first comment (e.g. I think...). At the same time, a separate process manages access to databases needed (WordNet, TriGram...) to display the main comments in a current conversation. This system allows us to simulate the human's speed of typing, but moreover we simulate human typing errors by using backspace and delete keys.

4. Demonstration

4.1. Flow chart



4.2. Example of interaction

Hello, I'm Elizabeth. How are you?
Fine thank you.

Good, I'm happy about that.
Does your family live in Australia?

Mmmm....
I have a small family. My family is in Europe.
Why are you in Australia?

I don't feel like speaking about that.
Have you a lot of friends?
Yes, I have friends.

Ok.
What's your religion?

I'm a Christian, but I don't like to speak about religion.
Are you going on holidays this year?

Mmm...
This year I 'm not going on holidays, because I 'm working in Australia. Generally, when I'm on holidays, I do nothing because I like sleep.

Where did you go on holidays last year?
Nowhere, I had no time.

No problem.
Would you like to come on holidays with me?

I'm so flattered.
Do you know Spanish?

I just know French but in this test, we must speak English.
Bye.
Bye.Nice to meet you.

5. Conclusion

To indicate our position in relation to the Loebner Prize, we think that this competition only answers a part of Turing's initial question, which is "Can machines think like a human?" and not "Can machines think?" (Turing).

We object to the fact that each judge has only a limited time to converse with each subject. This may not allow the judge to decide without hesitation if it's a machine or not. On the other hand for this competition, a cross-section of the community will be present. Our program needs luck to be able to converse with a speaker in a satisfying way. If a judge uses the pre-defined subjects in our system, the conversation will be coherent. However, we need to include a large number of pre-defined subjects. We also need to include systems of spelling correction which perform well, to try to retrieve keywords from words with spelling errors for instance. In fact we have only been working in this area for two months, we don't think that anyone could develop an intelligent machine so quickly. The future will tell. We can only wait and see...

6. References

Brill, E. A simple ruled-based part of speech tagger.

Brill, E, & Marcus, M.. Automatically acquiring phrase structure using distributional analysis.

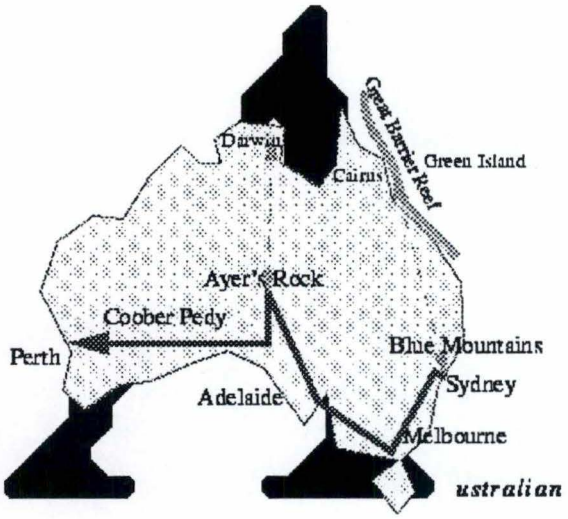
Golding, A.R. (1995) A baysian hybride method for context-sensitive spelling correction *Mitsubishi Electric Information Technology Center America.*

Golding, A.R. & Schabes, Y. (1996). *Combining trigram-based and feature-based methods for context-sensitive spelling correction.*

Powers, D.M.W. (1997) *Learning and application of differential grammars.*

Resnik, P. (1995) *Disambiguating nouns groupings with respect to WordNet senses.*

C. Pésentation ANLPF



NLPF

atural anguage rocessing ortnight

The Australian NLP community is pleased to announce that it will be hosting two international NLP conferences in January 1998, and has organized a fortnight of associated activities around the country to ensure that you are able to make the most of your trip to Australia.

The joint NeMLaP3 and CoNLL98 conference on New Methods in Natural Language Processing and Computational Natural Language Learning will be held in and around Sydney from January 12th to 17th 1997. The joint conference starts with a two day Tutorial Programme in the nearby Blue Mountains (Monday and Tuesday, 12-13 January) and the main conference consists of three days of Conference Programme with 30 submitted and invited presentations (Thursday to Saturday, 15-17 January).

There will be three associated workshops which will be held around Australia as part of the Australian Natural Language Processing Fortnight, and the 1998 Loebner Prize for Artificial Intelligence will also be held at Sydney's PowerHouse Museum on Sunday 11th January. The Human Computer Communication workshop will provide a bridge between the Loebner Prize competition and the NeMLaP/CoNLL conference and will include sessions in association with the Loebner Prize competition at the PowerHouse Museum (Sunday, 11 January) as well as workshop papers on the NeMLaP/CoNLL free day (Wednesday, 14 January). The workshop is also inviting demonstrations of NLP/HCC systems suitable for showing the general public the state of the art, and the PowerHouse museum will showcase HCC/NLP demos and Loebner Prize entries throughout the week.

The following week, the Australian Natural Language Postgraduate Workshop will be held in Melbourne (Monday and Tuesday, 19-20 January) and the workshop on Paradigms and Grounding in Language Learning will be held in Adelaide (Wednesday 21st January).

Finally, the annual Australian Computer Science Week (ACSW) follows in Perth - consisting of the Australasian Computer Science Conference and a number of specialist events (Theory, Computer Architecture, DataBase). Last year's ACSW program may be inspected here.

The NLP fortnight goes from Sunday 11th January to Saturday 24th January and will take you around most of Australia's major cities (including at least Sydney, Melbourne and Adelaide - Perth too if you plan on attending ACSW) and a multiday sightseeing trip to Ayer's Rock and the other desert sights is planned to immediately follow ANLPF before you head home, or to Perth for ACSW.

Organizers:

David Powers, Flinders Uni
Sandra Williams, Macquarie Uni/Microsoft Research Inst
Chris Manning, Sydney Uni
Dominique Estival, Melbourne Uni
Robert Dale, Macquarie Uni/Microsoft Research Inst
Ingrid Zukerman, Monash Uni
Peter Wallis, Defence Sci+Tech Org
Veronique Bastin, Notre Dame de la Paix,Namur/Flinders
Denis Cordier, Notre Dame de la Paix,Namur/Flinders

This webpage is http://www.cs.flinders.edu.au/research/AI/ANLPF/.

D. Règles du Loebner Prize

OFFICIAL RULES

1999 LOEBNER PRIZE COMPETITION IN ARTIFICIAL INTELLIGENCE

Additions and alterations to these rules since the last competition are shown in italic

1. The objective of the 1999 Loebner Prize Competition in Artificial Intelligence is to identify the computer system that can best succeed in passing a modern variant of the Turing Test. Judges will attempt to distinguish computer systems (referred to henceforward as "contestants") from one or more human beings (referred to henceforward as "confederates" or "human confederates") based on interactions with these computer entries. Judges may include children, people with disabilities, experts in Psychology, Linguistics or Artificial Intelligence, etc. and are intended to be in part representative of the community at large and in part of the Cognitive Science community.

2. Applicants may be individuals, organizations, businesses, schools, corporations, institutions, or other entities. Individuals need not have institutional affiliations.

Applicants may be of any nationality or age. Applicants may submit only one entry during any year.

3. If there are two or more entries there will be a Loebner Prize contest. The medal and cash award will go to the designer of the computer system with the highest score, based on the ratings of the contest judges.

4. If there is only one entry, Loebner Prize Medal and the $2000.00 cash award will be awarded to that entry. If there are no entries in the 1999 Loebner Prize Competition, the $2000.00 cash prize will be added to the cash award for 1999, making the award for 1999 $4000.00.

5. It is the task of the computer entries to respond to the communications of the judges in such a manner as to imitate the responses of a human. No constraints will be placed on the judges', confederates' or contestants' conversations. However, the competition is widely publicized, the transcripts will be published, and all participants must assume resposibility for their own contributions. As in any normal converstation, both participants have a role in determining the direction of the conversation and are free to decide want topics they do or do not wish to discuss. Entries must be prepared to communicate for an indefinite period of time.

6. Computer entries may contain standard or customized software and hardware. The system may be of any type as long as it contains no genetic material and as long as its replies are in no manner controlled by human beings or other organic systems in real time.

7. Entries will be required to run on hardware located at the competition site or an associated site. NO TELECOMMUNICATIONS WILL BE PERMITTED. Finalists chosen to participate may submit their entries as programs recorded on standard machine readable media (magnetic or optical storage) together with operating documentation. Macintosh, DOS/Windows and Unix computers with standard mass storage peripherals will be available on-site for the contest. Efforts will be made to secure the use of other standard equipment as required, but this cannot be guaranteed.

Where appropriate hardware cannot be provided by the organizers, entrants will be required to provide/arrange the hardware for operation at an agreed site. Personnel will be prepared to operate the computer entries according to documentation provided by the contestant.

8. Appropriate steps will be taken to prevent the unauthorized duplication or publication of contestants' entries, however neither the Cambridge Center, the Prize Committee, nor any of their agents can guarantee absolute security. Contestants, by entering this contest, imply their understanding of this and agree to hold the Cambridge Center, the Prize Committee and their agents harmless should there be any unauthorized duplication or publication of programs. Entrants who wish absolute security will be allowed to operate their entries and/or provide their own hardware on site. Entries may be used by the Prize Committee for demonstration or publicity purposes after the contest unless it is agreed otherwise prior to the contest.

9. Computer entries will be required to record the conversations as text files on magnetic media. The recordings will remain the property of the Cambridge Center, which will also retain the copyright on transcripts or other representations, magnetic or otherwise, of the recordings.

10. Applications must be accompanied by e-mail transcripts recording actual interactions between the system to be entered and one or more human beings. The protocols may not exceed 2,500 words.

11. The selection process may entail interaction between selection personnel and computer entries. Applicants will be notified of the selection decision by 15th November 1998, although this may be provisional contingent on appropriate arrangements being possible to accommodate unusual hardware. No more than eight entries will be selected as finalists to compete in a real- time and simultaneous contest to be held on Friday 21th January 1999 at The Flinders University of South Australia (date and location subject to change).

12. Judges will have one or more opportunities to interact with each of the computer terminals available concurrently during the contest. Judges will be allowed unrestricted communications. They will be informed that at least one of the terminals is controlled by human confederates and that at least two of the terminals are controlled by computers.

13. One or more referees may be present to limit communications of the confederates.

14. Judges will not be allowed to interact with each other and will be instructed to provide individual ratings of each of the appropriateness and responsiveness of the replies.

15. The names "Loebner Prize" and "Loebner Prize Competition" may be used by contestants in advertising only by advance written permission of the Cambridge Center. Advertising is subject to approval by representatives of the Loebner Prize Competition. Improper or misleading advertising may result in revocation of the prize and/or other actions.

16. Computer programs must communicate in the English language.

17. To avoid giving judges cues as to the nature of the programs, programs run from a shell should use the default I/O (stdin/stdout). Programs run in a windowed environment should use black on white in a 25x80 window or full screen mode, using the standard font, and should seek to mimic the confederate communications program (download or screendump possible): it displays a menu bar with Connection Font TTY View Help. Providing a View menu option to set the title (for the title bar of the window) is desirable but not obligatory. Since ideally all entries will be connected to the communications program over serial lines or (preferably) via a network, like the confederates, entries are advised to provide a commandline or Communication menu option to communicate via a serial line or TCP socket, but are again not obliged to do so.